UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

Departamento de Teoría de la Señal y Comunicaciones

IMPLEMENTACIÓN DE UN SISTEMA OFDM EN UN DISPOSITIVO SFF SDR

Autor: Carlos Valverde Muñoz

Tutora: Ana García Armada

Leganés, Noviembre de 2010

Título: IMPLEMENTACIÓN DE UN SISTEMA OFDM EN UN DISPOSITIVO SFF SDR

Autor: Carlos Valverde Muñoz

Tutora: Ana García Armada

EL TRIBUNAL

Presidente: María Julia Fernández-Getino

Vocal: Víctor P Gil Jiménez

Secretario: Enrique San Millán

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

Resumen

En los últimos años, el desarrollo en los sistemas de comunicaciones ha tenido un gran auge debido a las mejoras en los sistemas de procesado digital de señales. Entre ellos, destacan los sistemas basados en las técnicas de modulación OFDM con un gran crecimiento y una gran presencia en diferentes medios.

Las mejoras en esta técnica de transmisión de información son constantes gracias a una gran cantidad de proyectos de investigación. Por este motivo, surge el interés de llevar a la práctica una implementación hardware de un sistema de comunicación basado en OFDM.

En este proyecto, se han estudiado las características de los sistemas OFDM y se ha llevado a cabo un desarrollo de un sistema OFDM en código VHDL. De este modo, además de realizar un código general de un sistema OFDM que se implementa en módulos FPGA, se ha probado un transmisor OFDM en un dispositivo concreto pensado para las telecomunicaciones radio como es el “SFF SDR DP” de Lyrtech.

Así, conjuntamente al desarrollo del código, el proyecto profundiza en el estudio del dispositivo SFF SDR y el proceso necesario para configurarlo y programarlo para llevar a cabo pruebas hardware de sistemas de telecomunicaciones. Y al mismo tiempo, se ha comprobado la mejora de la eficiencia en cuanto a recursos que tiene lugar al programar estos dispositivos mediante VHDL con respecto a otras alternativas basadas en programación a alto nivel.

Abstract

In recent years, the development of communications systems has been boosted due to improvements in the digital signal processing. Among them there are systems based on OFDM modulation techniques with strong growth and a strong presence in different media.

Improvements in the techniques of transmission have been constant through a large number of research projects. For this reason it emerges an interest to produce a hardware implementation of a communications system based on OFDM.

In this project, we have studied the characteristics of OFDM systems and we have carried out a development of an OFDM system in VHDL code. Thus, in addition to a general code of an OFDM system that is implemented in FPGA modules, we tested an OFDM transmitter on a specific device designed for radio telecommunications such as the SFF SDR DP of Lyrtech.

Furthermore, the project extends the study of the "SFF SDR" devices in the process needed to set up and program it to execute tests of telecommunications systems hardware. And at the same time, it has been shown the improvement in the efficiency in resources that occurs when programming these devices using VHDL.

Índice general

Índice de figuras

Índice de tablas

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

Las comunicaciones digitales han tenido un crecimiento muy rápido en los últimos años. Este crecimiento ha sido posible gracias a mejoras en ciertas características como la robustez o la velocidad, que han aumentado las prestaciones de las comunicaciones.

De los sistemas con mayor auge en los últimos años, los basados en la técnica de modulación OFDM han sido de los que han tenido un mayor impacto. Aunque OFDM se comenzó a desarrollar por primera vez en la década de 1960, sólo en los últimos años, se ha reconocido como un método excelente para la comunicación de datos de alta velocidad.

La principal característica por la cual es muy utilizada la modulación OFDM es debido a su robustez frente al multitrayecto (multi-path), que es muy habitual en los canales de radiodifusión, frente a las atenuaciones selectivas en frecuencia y frente a las interferencias de radio frecuencia (RF).

Hoy en día podemos encontrar sistemas basados en OFDM en una gran variedad de sistemas de comunicación, entre los que destacan: las normas de la televisión digital terrestre o TDT (DVB-T), la radio digital (DAB), el protocolo de enlace ADSL, sistemas de transmisión inalámbrica como IEEE 802.11a/g/n (WiFi) o WiMAX y los sistemas de telefonía móvil de 4G (LTE).

Como consecuencia de este desarrollo, el estudio e implementación hardware de estos sistemas de comunicaciones tiene una gran proyección en el futuro de las telecomunicaciones. Sin embargo, la implementación hardware de este tipo de sistemas conlleva una cierta complejidad que suele suponer el uso de un lenguaje de programación de alto nivel, lo que acarrea una cierta ineficiencia en la utilización de los recursos hardware.

La motivación de este proyecto es llevar a cabo una implementación hardware de un sistema de comunicaciones pero utilizando un lenguaje de programación que utilice de manera eficiente los recursos disponibles. Para llevar a cabo esta implementación, se parte de un dispositivo con capacidad para el procesado digital de señales.

Dentro de la amplia gama de posibilidades existentes en el desarrollo de sistemas con procesado digital de señales, se ha escogido un sistema preparado para realizar el concepto de radio software (SDR) que integra entre otros módulos una FPGA.

Este dispositivo nos ofrece principalmente dos ventajas. La primera, poder trabajar de forma individual con cada uno de los módulos que lo componen, lo que nos proporciona un análisis del rendimiento. Y la segunda, que el desarrollo no se queda en la señal OFDM que una FPGA pueda obtener, si no que, además, nos permite implementar un sistema de comunicaciones de un mayor nivel gracias a que provee un módulo de conversión de datos y otro de radio frecuencia.

1.2 EXPERIENCIAS PREVIAS

Durante los últimos años en el Grupo de Comunicaciones del Departamento de "Teoría de la señal y Comunicaciones" se han llevado a cabo estudios de mejoras en los sistemas de comunicaciones OFDM. En la bibliografía, podemos ver en [1] y [2] algunos ejemplos.

Además, se ha buscado realizar la implementación de este tipo de sistemas de comunicación en diferentes trabajos. Ejemplos de estos trabajos son: "*Prototipado de un sistema WiMaX MIMO 2x2*" [3] o "*Descripción Hardware de Algoritmos de estimación de canal y sincronización en tiempo-frecuencia para un sistema 2x2 MIMO-OFDM*" [4].

Estos y otros trabajos se han implementado utilizando las técnicas de síntesis de MATLAB y Simulink, las cuales utilizaban prácticamente la totalidad de los recursos hardware. Mediante este camino se consiguieron llevar a cabo simulaciones que permiten evaluar este tipo de sistemas, pero no obtener una señal transmitida.

Por tanto, esta forma de implementación se ha mostrado incapaz de finalizar la programación de los dispositivos asociados y obtener así, una señal medible en un osciloscopio.

Por estos motivos se decidió el cambio de rumbo en el método de desarrollo de aplicaciones y en su implementación hardware, a los utilizados en el presente proyecto.

1.3 OBJETIVOS

El presente proyecto nace de la necesidad de implementar un sistema de comunicaciones OFDM en un dispositivo hardware, de forma que todos los estudios y mejoras que se llevan a cabo en el departamento, tengan un reflejo a nivel real de implementación.

Por tanto, el principal objetivo de este proyecto es el de conseguir obtener un sistema de comunicaciones OFDM en un dispositivo SFF SDR, de forma que se pueda analizar la señal transmitida es un osciloscopio.

Además, el método elegido, busca una mejora con respecto a experiencias previas, de la eficiencia en cuanto a los recursos hardware utilizados.

A continuación, se presenta una enumeración más detallada de los objetivos perseguidos por el proyecto:

- Realizar un estudio sobre el método de desarrollo de sistemas de comunicaciones en un dispositivo SFF SDR.

- Configurar un dispositivo SFF SDR para conseguir la transmisión de una señal OFDM paso banda.

- Llevar a cabo un estudio de las características esenciales de un sistema OFDM.

- Mejorar la eficiencia en la utilización de recursos de una implementación de un sistema de comunicaciones en un dispositivo SFF SDR mediante el desarrollo de código VHDL.

- Desarrollar un sistema de comunicaciones OFDM completo, sintetizable y programable en módulos FPGAs.

- Simular el correcto funcionamiento del sistema completo.

- Medir y analizar la salida del sistema transmisor en un laboratorio de comunicaciones.

1.4 CONTENIDO DE LA MEMORIA

El contenido de la memoria se estructura en los siguientes capítulos:

- En el capítulo 2, “Herramientas de desarrollo”, se describen el entorno de trabajo tanto hardware como software utilizados. En la parte hardware, analizamos el dispositivo SFF SDR detallando las características más relevantes de cada uno de los módulos que lo componen.

  En la parte software examinamos, de las posibles herramientas de programación del dispositivo SFF SDR, las que se han escogido según el tipo de programación.

- En el capítulo 3, “OFDM”, se revisan los principios teóricos de un sistema de comunicaciones OFDM tratados en el presente proyecto.

- En el capítulo 4, “Diseño y desarrollo de un sistema de comunicaciones OFDM” se explica todo el proceso seguido al desarrollar el sistema de comunicaciones en lenguaje VHDL.

  El desarrollo de parte está compuesto de un análisis de necesidades, un diseño, y una simulación. Tratándose, primero los bloques de que se compone el sistema individualmente, y posteriormente, su integración para formar un modulador y un demodulador.

- En el capítulo 5, “Implementación en un dispositivo SFF SDR” se detallan los pasos necesarios para integrar el bloque transmisor en el dispositivo SFF SDR, así como la configuración necesaria del mismo para obtener una señal OFDM medible. Por último, se analizan los resultados obtenidos.

- En el capítulo 6, “Conclusiones” se establecen las conclusiones sobre el desarrollo del presente proyecto.

- En el capítulo 7, “Historia del proyecto” se explican las diferentes fases que ha tenido el desarrollo del proyecto, analizando las tareas realizadas, los problemas más relevantes encontrados, y los resultados al final de cada una de las fases.

- En el capítulo 8, “Líneas Futuras” se establecen posibles mejoras que pueden aplicar del sistema desarrollado.

- En el Apéndice A, “Presupuesto”, se especifica el presupuesto asociado a la realización del presente proyecto, teniendo en cuenta tanto los costes asociados al personal, como los costes directos de material y los costes indirectos.

- En el Apéndice B, “Guía para generar y cargar programas en el dispositivo SFF SDR”, se detallan los pasos necesarios para poder configurar y programar el dispositivo utilizado.

2 HERRAMIENTAS DE DESARROLLO

2.1 HARDWARE

2.1.1 Introducción

En este capítulo se describe la plataforma hardware utilizada en las pruebas del proyecto. Toda la información expuesta proviene de los manuales de los fabricantes [5-12].

Para la implementación del proyecto se utilizó un sistema SDR ("Software Defined Radio)". Los sistemas SDR son sistemas de radio comunicación en los que la mayoría de los elementos hardware típicos de un sistema de comunicaciones (mezcladores, filtros, amplificadores, moduladores/demoduladores) son llevados a cabo mediante una implementación software.

SDR ofrece por tanto una arquitectura flexible, la cual permite mejorar e incrementar el desarrollo de una manera sencilla modificando el software. Esto permite, una gran versatilidad para el desarrollo de diferentes sistemas de transmisión de datos y estándares.

Así, las principales ventajas del sistema por el que se escogió para la implementación del proyecto, las podríamos resumir en:

- Versatilidad
- Sistema Compacto
- Eficiencia
- Fácil de mejorar

2.1.2 Lyrtech SFF SDR DP

El sistema utilizado es el SFF SDR DP: Small Form Factor Software Defined Radio Development Platformes, desarrollado por la compañía canadiense Lyrtech. La plataforma esta concebida para el diseño y desarrollo de aplicaciones de radio e incorpora módulos de las empresas Xilinx y Texas Instruments.

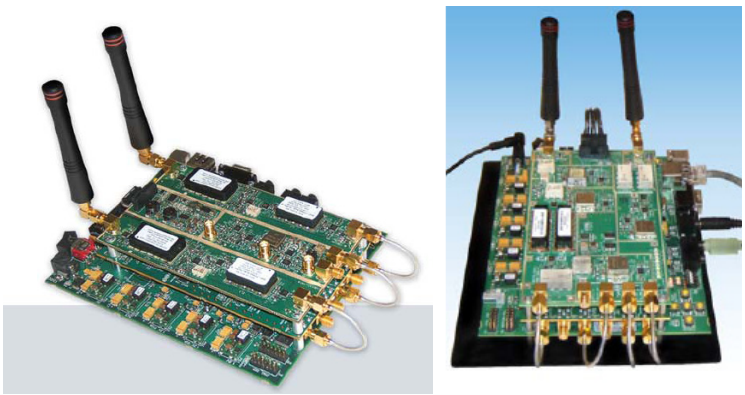


Figura 2.1: *Imágenes de la plataforma de desarrollo*

Está compuesta de tres módulos como podemos ver en la figura 2.2. Los tres módulos son:

- Modulo de Procesado Digital (Digital Processing Module)
- Modulo de Conversión de Datos (Data Conversion Module)
- Modulo de Radiofrecuencia (RF Module)

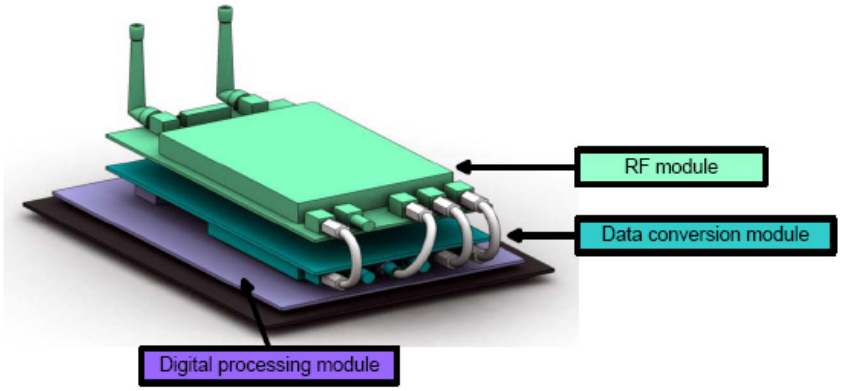


Figura 2.2: *Módulos de la plataforma de desarrollo*

El diagrama de bloques del sistema que forma los tres módulos es el siguiente:

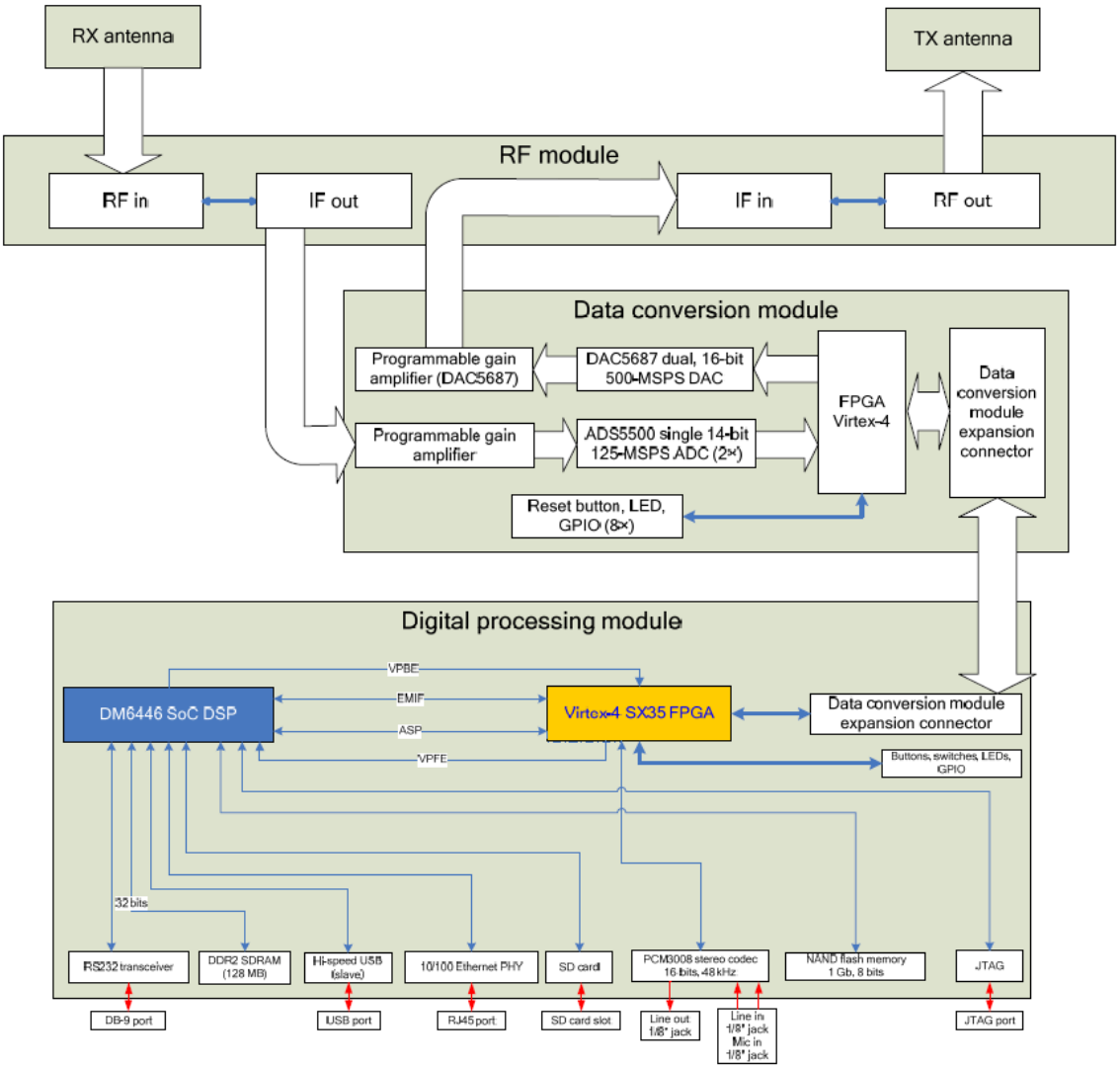


Figura 2.3: *Diagrama de bloques de la plataforma SFF SDR DP*

2.1.2.1 Módulo de Procesado Digital

El módulo llamado de procesado digital está compuesto por dos dispositivos: un DSP (procesador digital de señales o “Digital signal processor” en inglés) y una FPGA (del inglés “Field Programmable Gate Array”). En ambos se puede llevar a cabo la implementación del procesado de señal.

El dispositivo DSP es el modelo TMS320DM6446 (también llamado plataforma DaVinci) de “Texas Instruments”. A este tipo de dispositivos se les conoce como “System on Chip” (SoC) que quiere decir que toda la funcionalidad del dispositivo se programa directamente sobre el mismo chip. El chip está compuesto de un procesador ARM9 de “propósito general” (GPP:general-purpose processor) y un procesador TMS320C64x que actúa de núcleo. El procesador ARM9 trabaja a una frecuencia de 594 MHz y el núcleo a 297 MHz. El DSP también contiene algunas memorias como 64 registros de 32 bits, memoria RAM para datos y programas, y acceso a memoria externa.

La FPGA es del tipo Virtex-4 XC4SV35 proporcionada por Xilinx. Sirve como coprocesador y es quien controla todas las interfaces de entrada/salida de la plataforma. La Virtex-4 tiene 96x40 bloques lógicos configurables (CLBs).

Se puede llevar a cabo una comunicación entre el DSP y la FPGA a través de un protocolo proveniente de la modificación del protocolo VPSS [7] (“Video Processing Subsystem protocol”). Consiste en la interfaz VPFE (Video Processing Front End) usada para transmitir de datos de 16 bits de resolución a 75 MHz de la FPGA a la entrada del DSP, y en la interfaz VPBE (Video Processing Back End) que se encarga de los datos de salida del DSP y de entrada a la FPGA, con la misma resolución y a 75 MHz.

2.1.2.2 Módulo de Conversión de Datos

En la siguiente figura, vemos el esquema del módulo de conversión de datos:

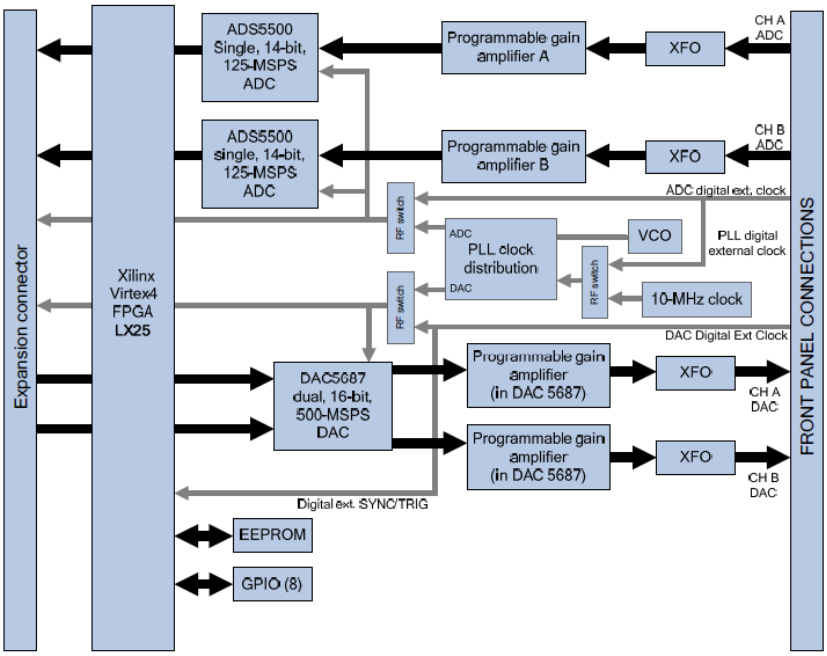


Figura 2.4: *Esquema del módulo de conversión de datos*

Este módulo tiene los conversores análogico-digital (ADC) y digital-analógico (DAC). El DAC es el modelo DAC5687 de Texas Instruments que tiene dos canales que pueden operar simultáneamente con una resolución de 16 bits. El ADC es del tipo ADS5500 también de Texas Instruments, y puede muestrear a una tasa de 125 Mega-muestras por segundo (MSPS) con una resolución de hasta 14 bits.

En este modulo además hay otra FPGA, en este caso un modelo Virtex-4 XC4VLX25 de Xilinx, que se encarga de ajustar los valores de configuración del conversor de datos. Esta FPGA no tiene el propósito de ser reprogramada y únicamente se pueden acceder a los parámetros de configuración a través del DSP del módulo de procesado digital.

Se puede seleccionar el reloj de trabajo del conversor de datos, pudiendo utilizar una fuente externa, el reloj de referencia o un VCO ("Voltage Controlled Oscillator").

Además este módulo, usa un PLL ("Phase-Locked Loops" o lazos de seguimiento de fase) para mantener constante la fase y la frecuencia.

2.1.2.3 Módulo de Radiofrecuencia

El módulo de radio frecuencia está dividido entre la parte del transmisor y la del receptor.

A) Transmisor

En la figura 2.5 vemos un esquema del transmisor. El transmisor recibe las señales de fase y cuadratura, y las mezcla con la portadora, transmitiendo el resultado por la antena. Las señales en fase y cuadratura son filtradas paso-bajo y mezcladas con la onda portadora por separado. La señal de cuadratura es mezclada con una versión invertida de la señal portadora. Se pueden elegir dos bandas de transmisión: 262-438 MHz o 523-876 MHz, y un PLL se encarga de mantener estable la frecuencia.

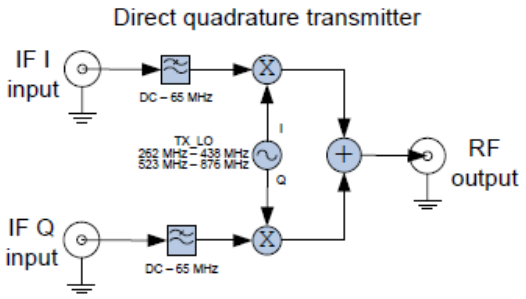


Figura 2.5: *Esquema del transmisor del módulo RF*

B) Receptor

La otra parte del módulo de radiofrecuencia es el receptor de tres etapas superheterodino que produce una señal IF (Frecuencia Intermedia) de 30 MHz. La FI se obtiene de la mezcla de la señal proveniente de la antena filtrada en un rango de 20 a 1000 MHz, con el oscilador local. El receptor es de tres etapas ya que la señal se mezcla en tres etapas como vemos en el esquema del receptor (figura 2.6). La primera mezcla es después del primer filtrado paso-bajo, y su frecuencia se puede escoger entre 1600 y 2500 MHz. El siguiente filtro es un paso-banda de 20 MHz de ancho de banda centrado alrededor de los 1575 MHz. Después, hay un mezclador fijo a 1275 MHz seguido de un filtro centrado a 300 MHz, y su ancho de banda puede ser elegido desde los 5 hasta los 20 MHz. Por último, es un mezclador con una frecuencia fija de 300 MHz seguido de un filtro paso-bajo. El resultado es una señal IF con una frecuencia de portadora de 30 MHz.

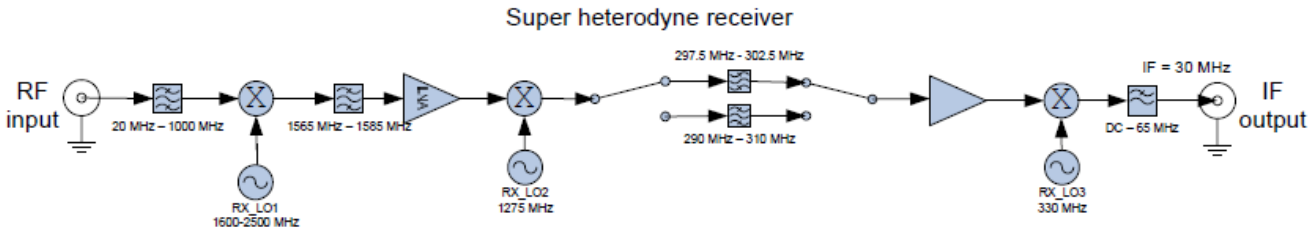


Figura 2.6: *Esquema del receptor del módulo RF*

2.1.3 Conclusiones Hardware

Con el fin de tener una idea general de las posibilidades que nos ofrece el hardware explicado en los puntos anteriores, llevamos a cabo un breve resumen con las características más importantes.

- TMS320DM6446 “digital media processor” (DMP) “system-onchip” (SoC) de Texas Instruments
- Virtex-4 SX35 FPGA de Xilinx
- Tasa máxima de muestreo de 125 MSPS de 16-bits en el DAC y de 14-bits en el ADC
- Frecuencias de trabajo del módulo de RF seleccionable en los intervalos de 200-930 MHz en el transmisor y de 20 a 1000 MHz en el receptor.
- Ancho de banda máximo de señal a transmitir de 65 MHz.

2.2 SOFTWARE

2.2.1 Introducción

El módulo “SFF SDR EVM/DP” puedes usarse mediante el desarrollo de aplicaciones tanto para la FPGA como para el DSP. Para los cuales, se utilizan dos modos de desarrollo: “Board Software Development Kit” (BSDK) y “Model-Based Development Kit” (MBDK).

BSDK proporciona al usuario la posibilidad de utilizar todas las funciones disponibles del sistema mediante el desarrollo de código en C/C++ o ensamblador para el DSP, o código HDL para la FPGA. Para ello, el usuario necesita un conocimiento de las interfaces del sistema como los módulos de conversión de datos o de radio frecuencia.

En el otro lado, el paquete de desarrollo MBDK proporciona librerías de Lyrtech para controlar las interfaces mediante bloques Simulink de MATLAB. Estos programas se encargarán de generar el código ejecutable que se carga en la FPGA y el DSP.

El objetivo de este proyecto era la implementación de un sistema de comunicaciones OFDM en un dispositivo SFF SDR, por lo que ambas opciones se podrían considerar validas a priori. Sin embargo, la programación mediante el desarrollo de un código VHDL mejora la eficiencia a la síntesis a través de MATLAB, por lo que en nuestro caso utilizaremos una implementación modo BSDK.

Por tanto, necesitaremos el entorno de desarrollo necesario para programar un sistema de comunicaciones OFDM en la FPGA del módulo de procesado digital. Pero como además los otros dos módulos (conversión de datos y RF) se controlan a través del DSP, necesitaremos también desarrollar una aplicación en C/C++ para programar el DSP.

Por este motivo en los siguientes puntos, además de analizar el lenguaje VHDL, veremos el entorno de desarrollo empleado para su integración en el módulo SFF SDR.

2.2.2 Entorno de desarrollo

Como ya comentamos, seguiremos la línea de desarrollo BSDK para llevar a cabo la implementación, para lo cual, es necesarios el siguiente software proporcionado por el fabricante:

- **Software de desarrollo para FPGA**
  - Xilinx ISE Foundation 9.2i
  - Xilinx Synthesis Technology (XST) 9.2i
- **Software de desarrollo para ARM/DSP**
  - Texas Instruments Code Composer Studio 3.3
- **Comunicaciones con el sistema**
  - “Lyrtech development tools”

Además de este software, se ha utilizado el programa **ModesIsim 6.2c** para llevar a cabo las simulaciones del código VHDL.

En los siguientes puntos veremos los entornos de desarrollo utilizados, profundizando en el lenguaje VHDL [13] [14].

2.2.2.1 VHDL

VHDL es una de las herramientas fundamentales en la realización de equipos electrónicos digitales, tanto para aplicaciones comerciales como para industriales y aeroespaciales. Su importancia ha crecido tanto en los últimos años que actualmente, es un elemento clave en cualquier metodología de desarrollo de circuitos.

VHDL viene de VHSIC ("Very High Speed Integrated Circuit") y HDL ("Hardware Description Language"). Es un lenguaje definido por el IEEE que se utiliza para describir circuitos digitales. Es por tanto, un lenguaje de descripción hardware, y como consecuencia, cubre unas necesidades distintas a las que dan servicio los lenguajes de programación habituales. Los HDL´s se diferencian de otros lenguajes en que permiten la descripción de cualquier sistema formado por un conjunto de procesos que se ejecutan en paralelo. Por lo tanto, una de las mayores ventajas de este lenguaje de programación es su versatilidad.

A pesar de ello, VHDL se emplea principalmente en la especificación, diseño y síntesis de circuitos electrónicos. El flujo de diseño es sencillo, una vez descrito el circuito con VHDL, se comprueba la sintaxis. Luego, se simula el comportamiento del mismo ante la aparición de unos estímulos en las entradas. Finalmente, un programa de síntesis convertirá el texto VHDL en una relación de componentes interconectados entre si, el "layout". Con el *layout* ya definido, es posible introducir el circuito en un chip programable como una FPGA, de forma que se comporte como el programa sintetizado a partir del código VHDL.

En el siguiente diagrama podemos ver el proceso completo que se lleva a cabo en el diseño de sistemas basados en lógica programable como VHDL.

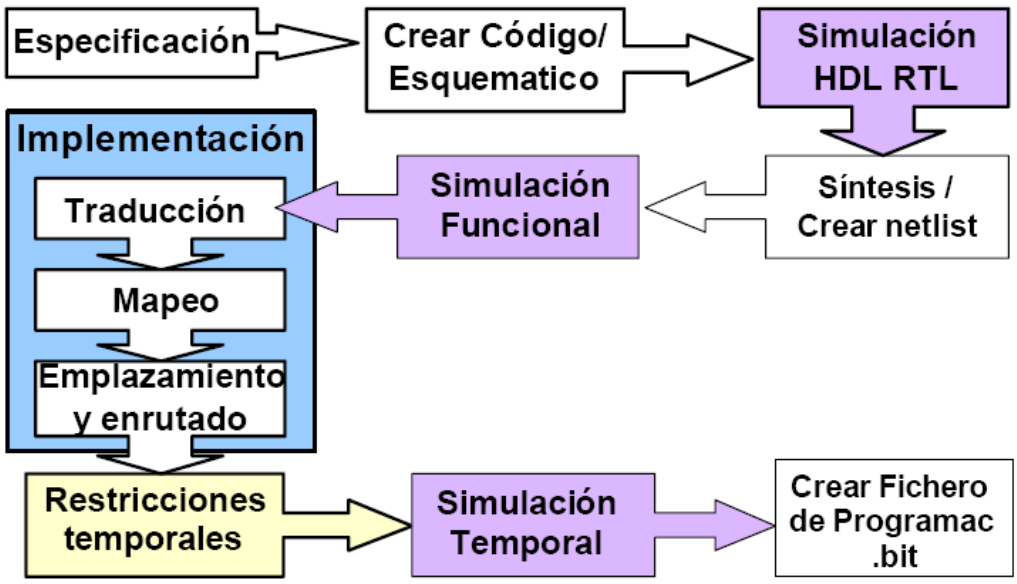


Figura 2.7: *Diagrama de desarrollo en VHDL*

2.2.2.2 Xilinx ISE

Para llevar a cabo el desarrollo del código VHDL, utilizamos la herramienta de desarrollo Xilinx ISE 9.2i [15] que proporciona todas las herramientas necesarias para la implementación y síntesis del código VHDL en la FPGA. Xilinx ISE proporciona la tecnología *SmartCompile* que sintetiza con una alta eficiencia temporal, y que además proporciona librerías de los módulos más comúnmente utilizados.

En la siguiente captura, vemos la interfaz de trabajo.

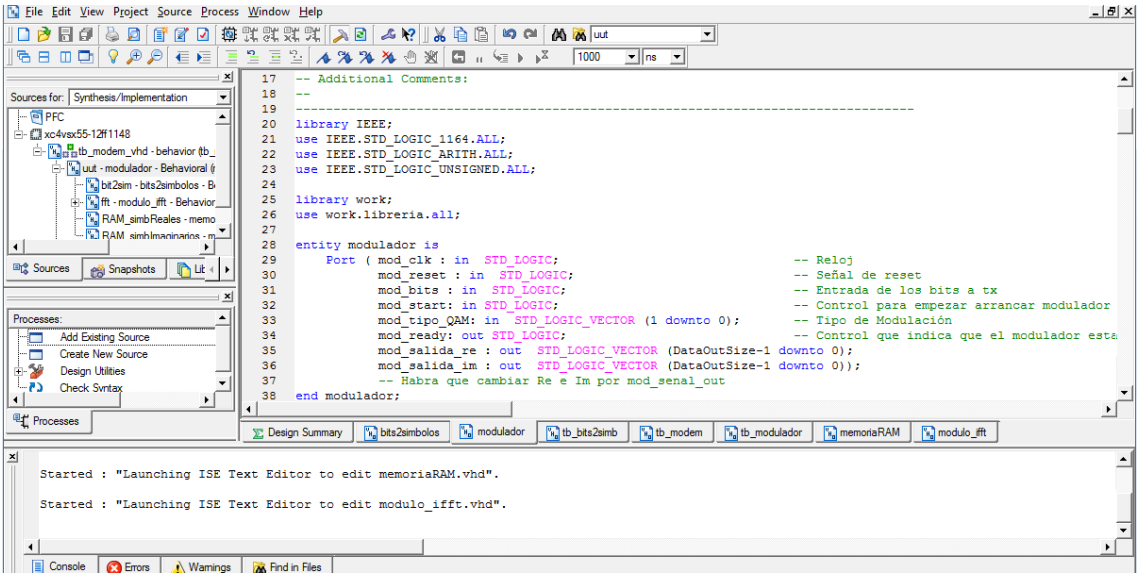


Figura 2.8: *Entorno Xilinx ISE*

El programa nos permite llevar a cabo las acciones de compilar, sintetizar y generar un fichero bit, con la información de nuestro código VHDL, que es el que se cargará en la FPGA [16].

2.2.2.3 ModelSim 6.2

Aunque Xilinx ISE nos proporciona un simulador propio, para el desarrollo de las simulaciones de los módulos implementados, utilizaremos el software Modelsim 6.2c. El motivo de utilizar este programa, es su mayor robustez, así como, una interfaz de simulación más clara, y el hecho de tener un mayor conocimiento sobre el mismo. Este programa se integra directamente sobre el entorno de Xilinx, como vemos en la siguiente imagen.

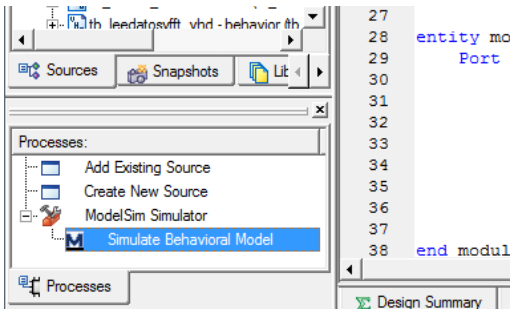


Figura 2.9: *Integración de Modelsim en Xilinx ISE*

El programa Modelsim, trae incorporado un entorno donde analizar los cronogramas (figura 2.10) que utilizaremos a lo largo de todo el proyecto.

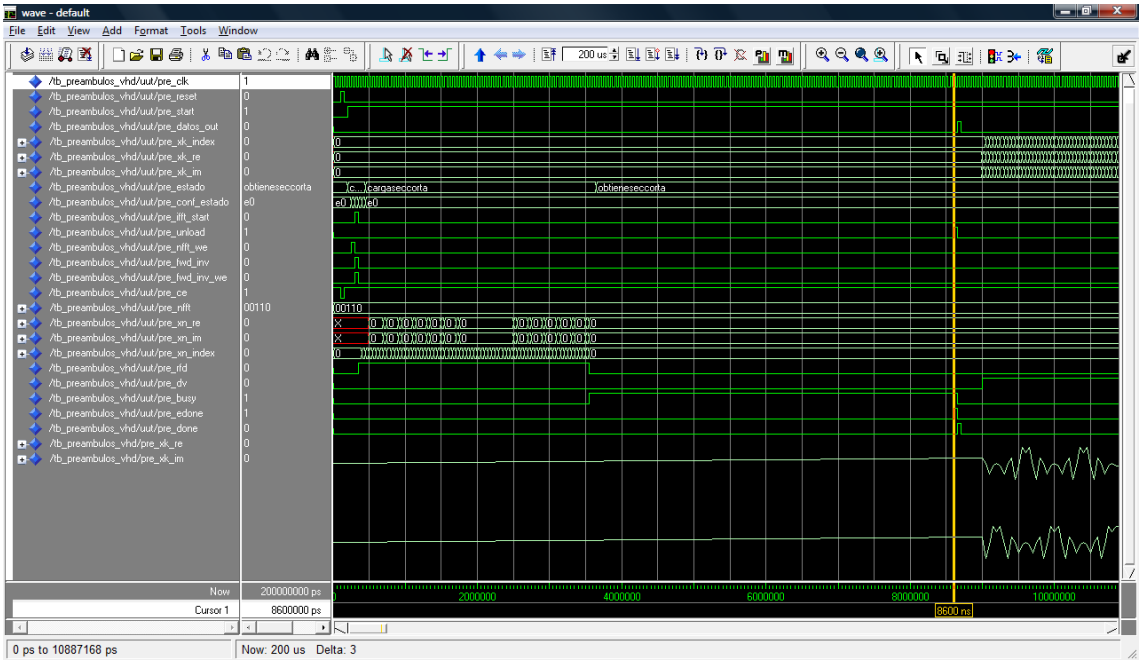


Figura 2.10: *Entorno de Modelsim 6.2c*

2.2.2.5 Code Composer Studio v3.3

Code Composer Studio (CCS) v3.3 [17] es una versión de CCS en la que se pueden programar una amplia variedad de plataformas. Soporta los códigos C, C++ o ensamblador para el desarrollo de aplicaciones.

El CCS es un entorno de desarrollo para la implementación de aplicaciones en módulos DSP/ARM. Incluye un compilador para la familia de dispositivos de Texas Instruments, un editor de código, un depurador, un simulador y demás funcionalidades necesarias para implementar aplicaciones.

CCS 3.3 además incluye un sistema operativo en tiempo real llamado DSP/BIOS, que incluye instrucciones para simular y soporta depuración mediante sondas JTAG.

Soporta los siguientes procesadores: TMS320C6000, TMS320C5000, TMS320C2000, TMS470, TMS570, ARM 7, ARM9, ARM 11, ARM Cortex M3, ARM Cortex R4, ARM Cortex A8, MSP430, TMS320F28 y TMS320F24x.

En la figura 2.11 podemos ver el entorno gráfico de desarrollo del Code Composer Studio 3.3.

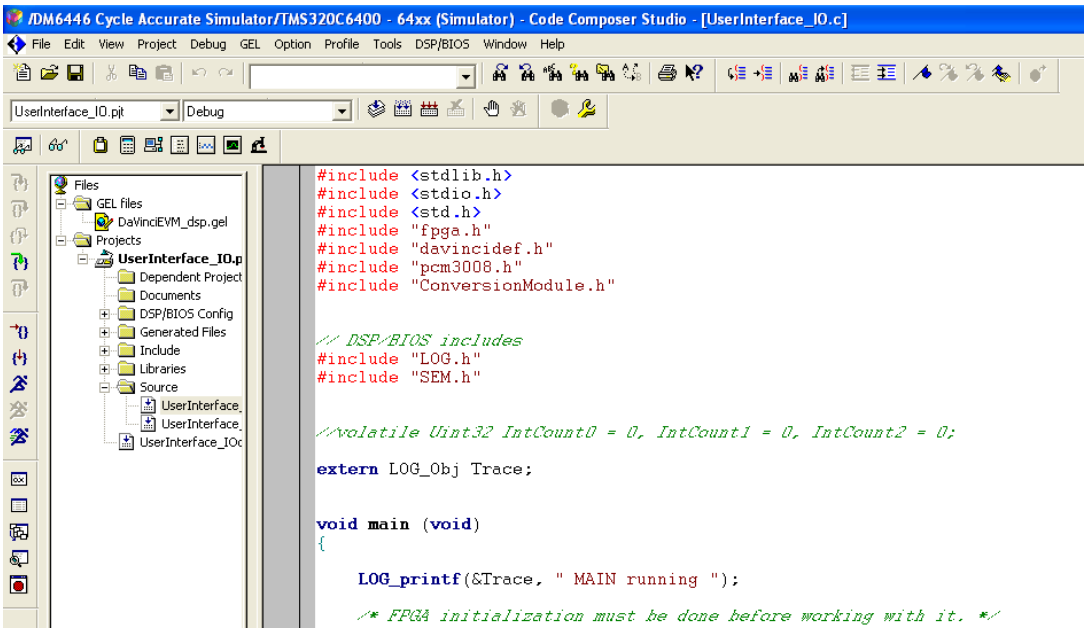


Figura 2.11: *Entorno CCS 3.3*

2.2.2.6 Lyrtech Development Tools

Para comunicarse con el dispositivo SFF SDR, el fabricante Lyrtech proporciona un software específico, mediante una serie de programas.

Debido a que en el presente proyecto, programamos el DSP y la FPGA mediante la generación de archivos ejecutables, de todas las herramientas proporcionadas, sólo utilizaremos la consola de comandos “Command Shell”.

Se trata de una consola de comandos, en la que, mediante las instrucciones del manual [18], podremos cargar los archivos ejecutables tanto en la FPGA como en el DSP.

3 OFDM

3.1 INTRODUCCIÓN

OFDM (Orthogonal Frequency Division Multiplex) [19-26] consiste en una multiplexación en frecuencia de diferentes portadoras, donde cada una transporta una información modulada siguiendo una constelación M-QAM o M-PSK.

El resultado es una señal que se transmite en paso banda, y que contiene a su vez N sub-bandas de transmisión pertenecientes a una serie de portadoras ortogonales a una tasa baja. De manera que, al transmitir todas a la vez, se consigue una tasa mucho más alta. Cada una de estas portadoras se comporta como un canal independiente que sólo sufre atenuación y no se produce dispersión en cada subcanal.

El uso de portadoras ortogonales entre sí, permite un mejor aprovechamiento de la banda de transmisión. En la FDM convencional la separación entre subportadoras adyacentes es como mínimo de 2/T, mientras que en OFDM la separación es de 1/T, que es el mínimo para que las subportadoras adyacentes sean ortogonales, con lo que se mejora la eficiencia espectral, como podemos observar en las siguientes figuras:

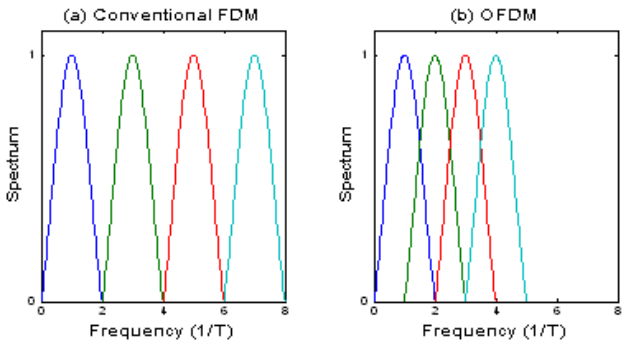


***Figura 3.1**: Espectro de las señales FDM y OFDM*

El significado de ortogonalidad en OFDM está basado en la idea de que cada subportadora tiene un número entero de ciclos durante el periodo de símbolo. Esto supone la ventaja de que cada subportadora tiene un nulo en el centro de la subportadora adyacente. El resultado es que la Interferencia Intersimbólica ("Inter Symbol Interference", ISI), idealmente es cero. Podemos expresar la ortogonalidad según el siguiente desarrollo matemático:

$$\frac{1}{T_{FFT}} \int_{t=0}^{T_{FFT}} e^{-i2\pi\Delta f(k-k')t}\, dt = \begin{cases} 1\ para\ k = k' \\ 0\ para\ k \neq k' \end{cases} \qquad (3.1)$$

Donde T_{FFT} denota la duración de un símbolo OFDM, Δf es la frecuencia entre dos subportadoras adyacentes y k y k' son el número de la subportadora para valores desde 1 hasta el número total de subportadoras.

El principal inconveniente que presenta la OFDM es la generación y detección del alto número de portadoras equiespaciadas que forman la modulación. Para mejorar el rendimiento las modulaciones y demodulaciones se realizan en tiempo discreto mediante la IDFT y la DFT respectivamente. Otra propiedad que complica el uso de transmitir mediante OFDM, es la sincronización que deben de tener emisor y receptor.

Por el contrario, la gran ventaja que presenta este sistema de transmisión, es su robustez frente a otras fórmulas en ciertas condiciones, como pueden ser la distorsión por atenuación en frecuencias altas en los cables metálicos y las interferencias y desvanecimiento o "fading" por multi-propagación.

3.2 DESARROLLO

3.2.1 Desarrollo matemático

El proceso para obtener la modulación OFDM se ilustra en la siguiente figura:

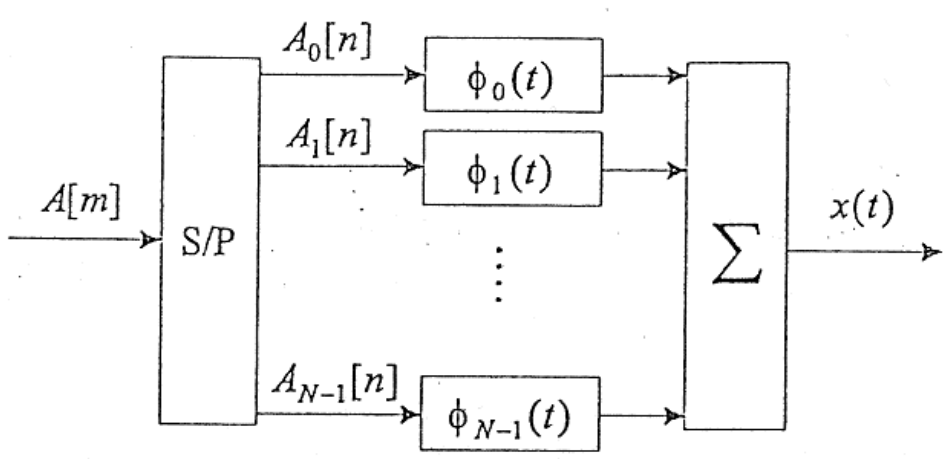


***Figura 3.2**: Proceso de modulación OFDM*

A[m] son el conjunto de símbolos a transmitir y el *x(t)* representado en la figura se obtiene a partir de la señal *s(t)* de la forma habitual, $\mathrm{x(t)} = \sqrt{2}\mathrm{Re}\{\mathrm{s(t)e^{jw_c t}}\}$ (3.2)

En este esquema, se emplea un conjunto de pulsos que se generan multiplicando un filtro prototipo por un conjunto de N portadoras distintas.

$$\mathrm{s(t)} = \sum_{\mathrm{n}} \mathrm{A^T[n]}\emptyset(\mathrm{t-nT}) = \sum_{\mathrm{n}} \sum_{\mathrm{l=0}}^{\mathrm{N-1}} \mathrm{A_l[n]}\emptyset_{\mathrm{l}}(\mathrm{t-nT}) \qquad (3.3)$$

Donde las funciones base $\Phi_l(t)$ son de la forma $\emptyset_{\mathrm{l}}(\mathrm{t}) = \frac{1}{\sqrt{\mathrm{T}}} \mathrm{e}^{\mathrm{j}\frac{2\pi \mathrm{lt}}{\mathrm{T}}} \cdot \mathrm{w_T(t)}$ (3.4) siendo $w_T(t)$ una ventana temporal rectangular de duración *T*.

Estas funciones base, forman una base ortonormal. Así, observamos como la OFDM en realidad es la superposición de N modulaciones paso banda que se transmiten simultáneamente.

Esta modulación descrita como hemos hecho, tiene el inconveniente de que su implementación práctica resulta difícil debido a que es necesario generar N portadoras complejas (2N reales) perfectamente enganchadas en fase. Si esta condición no se cumple, las funciones base dejan de ser ortogonales y aparece el efecto conocido como Interferencia entre portadoras ("InterCarrier Interference", ICI). Para evitarlo, muestreamos *s(t)* con periodo T/N obteniendo:

$$s[\mathrm{m}] = \sum_{l=0}^{N-1} \mathrm{A_l}\,[0]\emptyset_\mathrm{l}\left(\frac{\mathrm{mT}}{\mathrm{N}}\right) = \frac{1}{\sqrt{\mathrm{T}}}\sum_{l=0}^{\mathrm{N}-1} \mathrm{A_l}\,[0]\mathrm{e}^{-\mathrm{j}\frac{2\pi\mathrm{lm}}{\mathrm{N}}};\mathrm{m} = 0,\ldots,\mathrm{N}-1 \qquad (3.5)$$

Donde podemos comprobar que el término de la derecha, no es más que la DFT inversa de la secuencia A_l multiplicada por un factor constante.

$$DFT_N^{-1}\colon\ x[n] = \frac{1}{N}\sum_{m=0}^{\mathrm{N}-1} X(\mathrm{m})\mathrm{e}^{-\mathrm{j}\frac{2\pi\mathrm{nm}}{\mathrm{N}}} \qquad (3.6)$$

Por tanto, aprovechando además la eficiencia que nos prestan los algoritmos que calculan la transformada de Fourier en tiempo discreto, como la FFT, llegamos a la implementación del sistema modulador.

3.2.1.1 Modulador OFDM

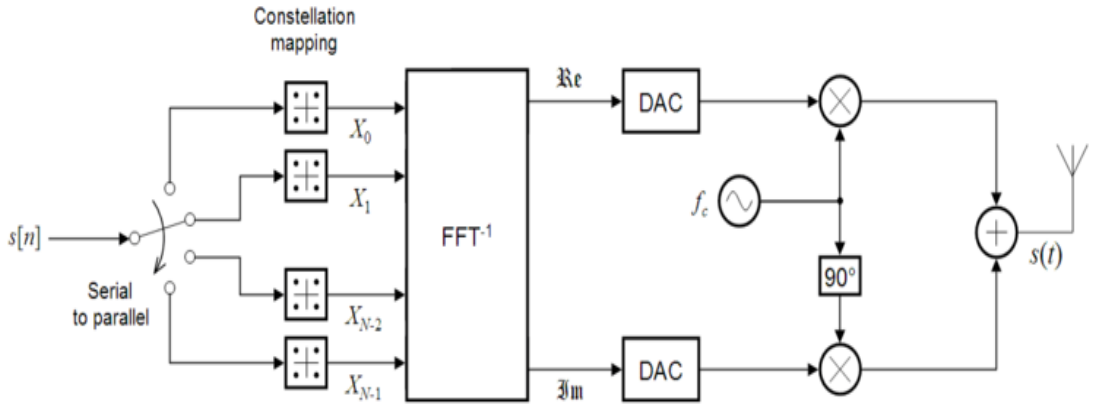


Figura 3.3: *Diagrama de bloques de un modulador OFDM*

Los bits de información que llegan al modulador, se separan en bloques paralelos que irán asociados a las diferentes portadoras. Cada bloque de bits se asocia según el tipo de constelación empleada (QPSK, QAM, etc.). Los N símbolos resultantes son modulados mediante el uso de la transformada inversa de Fourier, que obtiene dos señales (una real y otra imaginaria) que es la información modulada. Para la transmisión de las señales, se convierten al tiempo continuo mediante un conversor digital-analógico y se modulan en la banda de RF ambas señales en fase y cuadratura.

El sistema de recepción, sigue los pasos inversos a cada bloque, como muestra la figura 3.4.

3.2.1.2 Demodulador OFDM

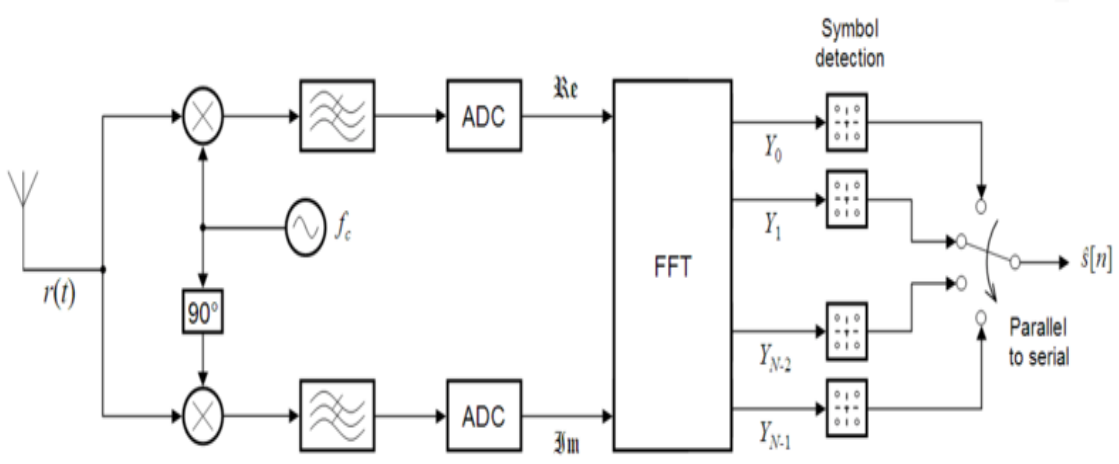


Figura 3.4: *Diagrama de bloques de un demodulador OFDM*

3.2.2 Estudio Espectral

Como ya se ha comentado, los sistemas OFDM, se basan en la multiplexación en frecuencia, es decir, a la superposición de diferentes señales moduladas con portadoras a diferentes frecuencias.

En la figura 3.5 vemos el espectro de cada portadora, que es una sinc centrada en la frecuencia f_n. La superposición de varias sinc nos da un espectro prácticamente plano en el rango de frecuencias del sistema.

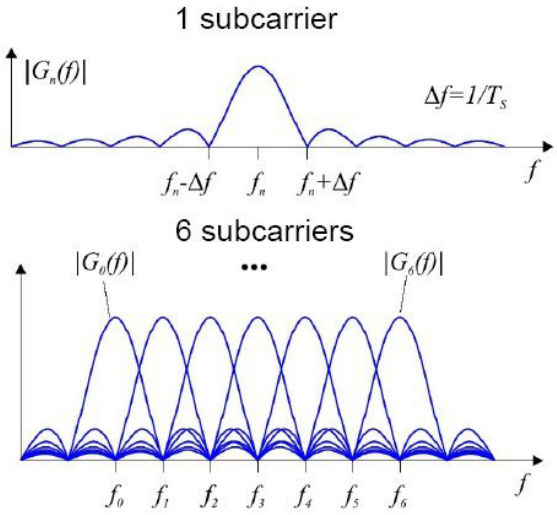


Figura 3.5: *Espectro de una señal OFDM*

Si los símbolos que se transmiten sobre cualquier par de subcanales están mutuamente incorrelados, podemos escribir el espectro como

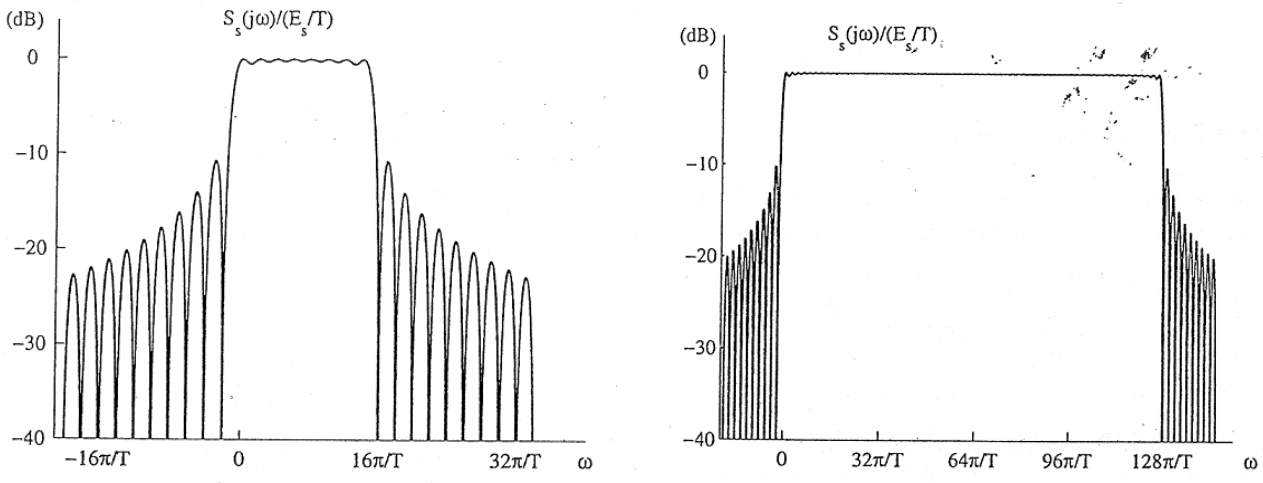
$$S_S(jw) = \frac{1}{T}\sum_{l=0}^{N-1} E_{s,l}\ |\Phi_l(jw)|^2 \qquad (3.7)$$

donde $E_{s,l}$ es la energía media por símbolo transmitido sobre el subcanal l-ésimo.

Sustituyendo $\Phi_l(jw)$ en la ecuación por su valor correspondiente, desarrollamos la expresión en frecuencia como sigue:

$$S_s(jw) = \frac{E_s}{T}\sum_{l=-\infty}^{\infty} \mathrm{sinc}^2\left(\frac{\left(w-\frac{2\pi l}{T}\right)T}{2\pi}\right) = \frac{E_s}{T}\mathrm{sinc}^2\left(\frac{wT}{2\pi}\right)\cdot\sum_{l=-\infty}^{\infty}\delta\left(w-\frac{2\pi l}{T}\right) \qquad (3.8)$$

En la siguiente figura [19] podemos ver el resultado del cálculo de la ecuación 3.8 para N=8 y para N=64.



$S_S(jw)$ N=8 *$S_S(jw)$ N=64*

Figura 3.6: *Espectro de una señal OFDM para N=8 y N=64*

Si llevamos a cabo el mismo estudio en tiempo discreto, llegamos a que si cada subcanal emplea la misma constelación con energía media *Es*, entonces

$$S_s(jw) = \begin{Bmatrix} E_s & |w| < \frac{N\pi}{T} \\ 0 & resto \end{Bmatrix} \quad (3.9)$$

Por lo que el espectro es plano en la banda del filtro transmisor, tal como podemos observar en la figura 3.7 [19].

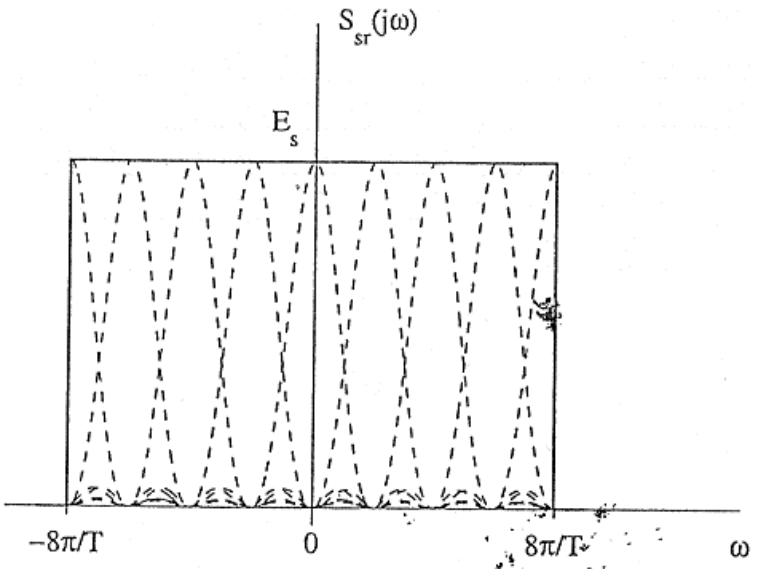


Figura 3.7: *$S_{Sr}(jw)$, espectro OFDM en tiempo discreto, N=8*

3.2.3 Prefijos cíclicos

Como se ha comentado, las señales OFDM tienen una buena respuesta en canales multi-path debido a que el uso de símbolos largos supone que apenas se vean afectadas por la interferencia entre símbolos (ISI). Además de la interferencia entre símbolos, podemos encontrarnos con interferencias provenientes de las subportadoras adyacentes en el propio símbolo. Es lo que conocemos como interferencia entre portadoras (Intercarrier Interference, ICI). La ICI, puede degradar significativamente la transmisión debido a la pérdida de ortogonalidad.

En la figura 3.8, podemos ver el efecto de la ICI en un canal que retrasa el primer símbolo interfiriendo con el segundo.

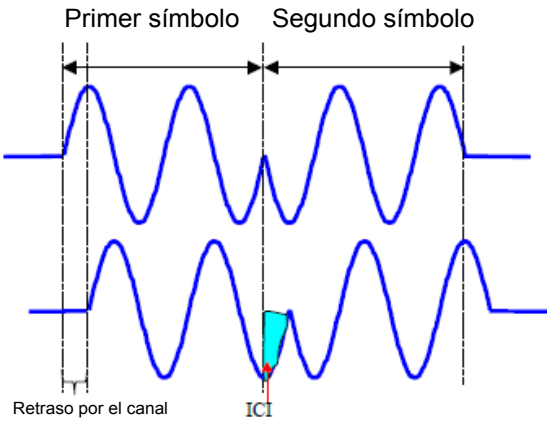


Figura 3.8: *Efecto de la ICI en los símbolos recibidos.*

Para minimizar, o eliminar este problema, se añade un prefijo antes del símbolo resultante de la IFFT. Este prefijo es obtiene de la parte final del símbolo resultante de la IFFT, garantizando de esta manera la periodicidad dentro del nuevo símbolo. Debido a esta característica de mantener la periodicidad se da el nombre de prefijo cíclico (CP del inglés “Cyclic Prefix”).

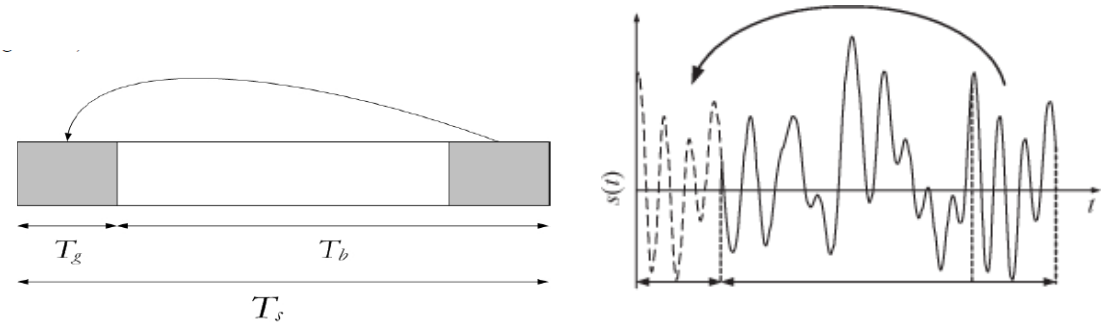


Figura 3.9: *Implementación del prefijo cíclico*
(a) *En trama* ***(b)*** *En señal*

Introduciendo el prefijo cíclico conseguimos que la señal obtenida como respuesta al canal sea una convolución circular en lugar de una lineal. Sólo la convolución circular tiene un efecto puramente multiplicativo en el dominio frecuencial, lo que evita la ICI.

Pero además, la incorporación del PC evita que un bloque interfiera con el siguiente, lo que se traduce en una ISI nula. Mientras la duración de la extensión sea mayor que la longitud del canal, el efecto de un bloque sobre el siguiente se limitará a corromper su extensión cíclica, pero no la parte de información.

Esto supone un pequeño cambio de la señal en su dominio en frecuencia. Esto ocurre debido a que la tasa de símbolo dentro del subcanal se ve reducida con el prefijo cíclico, manteniéndose la separación entre portadoras constante (Δf), lo que supone un estrechamiento de los lóbulos de cada subportadora, como vemos en la figura 3.9 obtenida de [26].

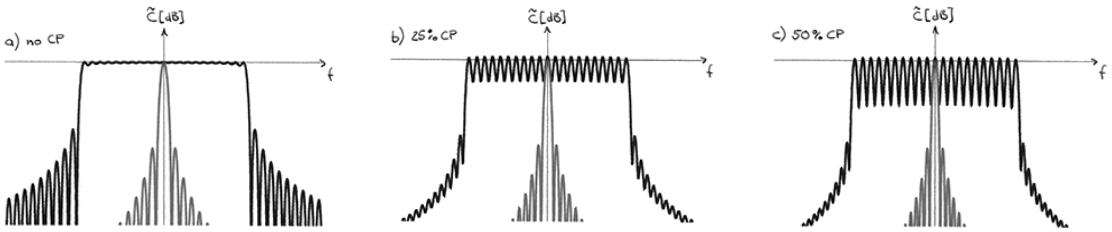


Figura 3.10: *Espectro OFDM para diferentes tamaños de PC*
(a) *PC=0* ***(a)*** *PC=0.25* ***(a)*** *PC=0.5*

Por último hay que hacer notar, que la inclusión del prefijo cíclico supone que las funciones base empleadas dejen de ser ortogonales, lo cual sólo indica la importancia de quitar el PC en el receptor.

3.2.4 Pilotos y bandas de guarda

En algunas aplicaciones de transmisión de datos que utilizan OFDM como técnica de transmisión, se emplean bandas de guarda y pilotos. El uso de bandas de guarda consiste en no transmitir información en las subportadoras OFDM de los extremos con el fin de reducir el espectro de transmisión. Esta técnica es muy útil si la limitación del canal está muy próxima al ancho de espectro, o si por el mismo canal se transmiten diferentes señales OFDM moduladas en banda, como muestra la figura 3.10.

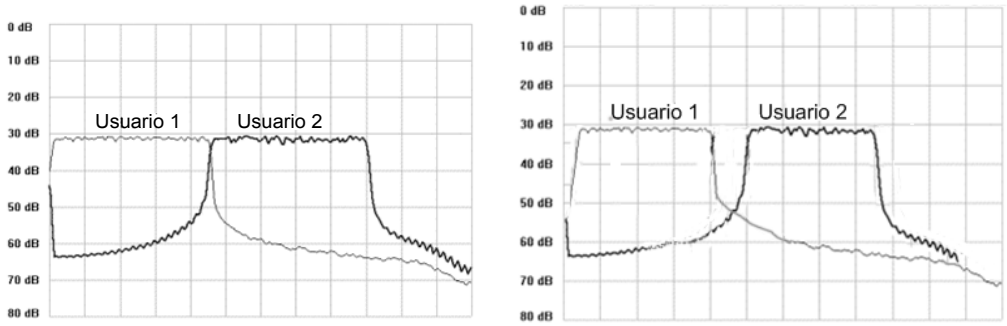


Figura 3.11: *Transmisión en banda OFDM en dos canales próximos*
(a) *Sin bandas de guarda* ***(b)*** *Con bandas de guarda*

Las subportadoras pilotos son utilizadas para transmitir un patrón conocido como referencia, y así minimizar los desplazamientos de frecuencia y fase producidos por el canal. Se utilizan repartidas por todo el espectro, para detectar los efectos del canal selectivos en frecuencia.

En la siguiente imagen podemos observar una posible configuración de las subportadoras, en las que ocho se dedican a la transmisión de pilotos y utilizan bandas de guarda.

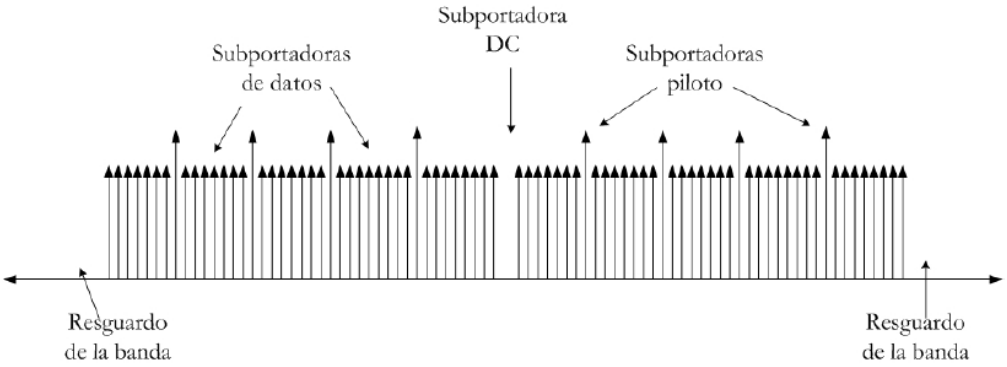


Figura 3.12: *Transmisión en banda OFDM en dos canales próximos*

3.2.5 Sincronización

La sincronización en tiempo y frecuencia entre el transmisor y el receptor OFDM es crítica en términos de prestaciones del sistema de comunicaciones. Una pequeña desviación en frecuencia entre el transmisor y el receptor provoca que las portadoras ya no sean ortogonales, causando una rotación de la señal en el dominio del tiempo, una reducción significativa de la amplitud de cada portadora y, el efecto más importante, la Interferencia entre Portadoras (ICI).

Hay diferentes estudios sobre algoritmos de sincronización en OFDM según el protocolo de transmisión utilizado. En este proyecto, nos centraremos en la estimación temporal obtenida de las pautas del estándar 802.11a [27], que pasamos a comentar en este punto.

En la primera parte veremos una breve descripción de las cabeceras descritas explicando más en detalle la forma de la secuencia corta de entrenamiento; y posteriormente, veremos la sincronización temporal compuesta de los bloques de detección de potencia y el algoritmo de sincronización.

3.2.5.1 Preámbulos

En el estándar 802.11a se definen las cabeceras llamadas PLCP que contienen información sobre las características de la señal OFDM transmitida y un conjunto de símbolos que ayudan en la detección y en la sincronización. La idea de las cabeceras es sincronizar el demodulador antes de ponerse a decodificar los datos que recibe. Las tramas de datos se llaman PSDUs y cuando se les añade las cabeceras se llaman PPDUs.

Los preámbulos del PLCP están compuestos por la repetición de diez veces la secuencia corta (“short training sequence”, STS) y de dos veces la repetición de la secuencia larga (“long training sequence” LTS).

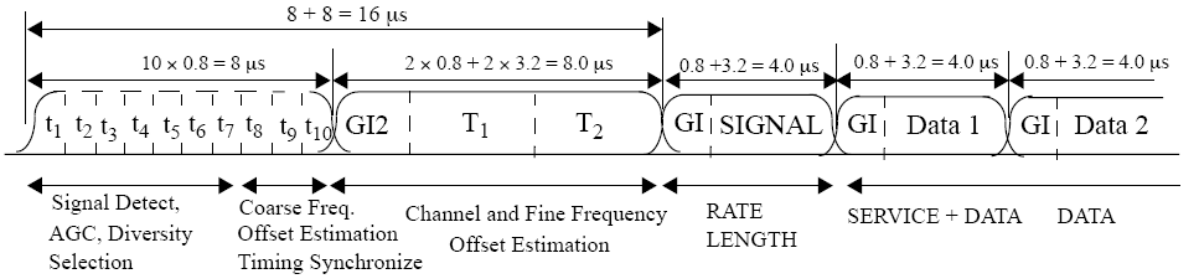


Figura 3.13: *Cabecera de una trama OFDM. 802.11a*

La figura 3.13 [27] muestra la cabecera de una trama OFDM descrita en el estándar 802.11a, donde t_i son los diez símbolos de la secuencia corta de entrenamiento y, T_1 y T_2 son los dos símbolos de la secuencia larga. El preámbulo del PLPC es seguido del campo SIGNAL y del campo DATA. En el estándar 802.11a está definida una frecuencia de muestreo de 50ns, con lo que en total, la longitud del entrenamiento es de 16 µs.

La secuencia corta se usa para la sincronización en el receptor, mientras que la secuencia larga se utiliza para estimar el canal y el offset de frecuencia. Como veremos en el punto 4.2.5**,** en desarrollo, nos centraremos en la sincronización temporal, con lo que pasamos a explicar en detalle la secuencia corta de entrenamiento.

Secuencia corta de entrenamiento

Para obtener la secuencia corta de entrenamiento, se modulan los símbolos de la secuencia S (3.10) [27] en 12 subportadoras.

$$S_{-26,26} = \sqrt{(13/6)} \cdot \{0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,$$
$$-1-j,0,0,0,+1+j,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,0,0,0,$$
$$+1+j,0,0,0,+1+j,0,0,0,+1+j,0,0,0,+1+j,0,0\} \qquad (3.10)$$

La multiplicación por el factor $\sqrt{13/6}$ es para normalizar la potencia media del símbolo OFDM obtenido, ya que se usa 12 de las 52 subportadoras definidas en el estándar. Esta secuencia hay que reordenarla para calcular la FFT.

Además, define que las entradas 38 a 63 se corresponden con los índices de la secuencia -26 a -1 y que las entradas 1 a 26 se corresponden con los índices 1 a 26, como vemos en la figura 3.13 [27].

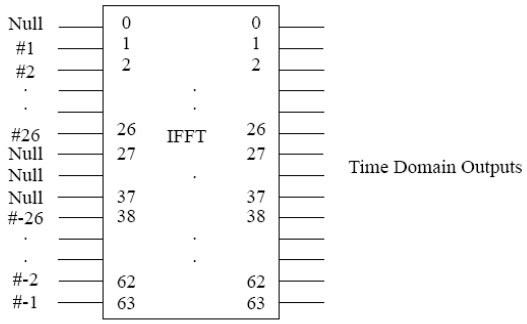


Figura 3.14: *Entradas y salidas de la IFFT*

Reordenando los índices de la IFFT según la figura 3.14 y asignando los símbolos de la secuencia S, obtenemos la tabla 3.1.

##	Re	Im	##	Re	Im	##	Re	Im	##	Re	Im
-32	0	0	**-16**	1.472	1.472	**0**	0	0	**16**	0	0
-31	0	0	**-15**	0	0	**1**	0	0	**17**	0	0
-30	0	0	**-14**	0	0	**2**	0	0	**18**	0	0
-29	0	0	**-13**	0	0	**3**	0	0	**19**	0	0
-28	0	0	**-12**	-1.472	-1.472	**4**	-1.472	-1.472	**20**	1.472	1.472
-27	0	0	**-11**	0	0	**5**	0	0	**21**	0	0
-26	0	0	**-10**	0	0	**6**	0	0	**22**	0	0
-25	0	0	**-9**	0	0	**7**	0	0	**23**	0	0
-24	1.472	1.472	**-8**	-1.472	-1.472	**8**	-1.472	-1.472	**24**	1.472	1.472
-23	0	0	**-7**	0	0	**9**	0	0	**25**	0	0
-22	0	0	**-6**	0	0	**10**	0	0	**26**	0	0
-21	0	0	**-5**	0	0	**11**	0	0	**27**	0	0
-20	-1.472	-1.472	**-4**	1.472	1.472	**12**	1.472	1.472	**28**	0	0
-19	0	0	**-3**	0	0	**13**	0	0	**29**	0	0
-18	0	0	**-2**	0	0	**14**	0	0	**30**	0	0
-17	0	0	**-1**	0	0	**15**	0	0	**31**	0	0

Tabla 3.1: *Símbolos de la secuencia corta de entrenamiento*

Haciendo la transformada inversa de Fourier (3.6) obtenemos los siguientes resultados:

##	Re	Im	##	Re	Im	##	Re	Im	##	Re	Im
0	0.046	0.046	**1**	-0.132	0.002	**2**	-0.013	-0.079	**3**	0.143	-0.013
4	0.092	0	**5**	0.143	-0.013	**6**	-0.013	-0.079	**7**	-0.132	0.002
8	0.046	0.046	**9**	0.002	-0.132	**10**	-0.079	-0.013	**11**	-0.013	0.143
12	0	0.092	**13**	-0.013	0.143	**14**	-0.079	-0.013	**15**	0.002	-0.132
16	0.046	0.046	**17**	-0.132	0.002	**18**	-0.013	-0.079	**19**	0.143	-0.013
20	0.092	0	**21**	0.143	-0.013	**22**	-0.013	-0.079	**23**	-0.132	0.002
24	0.046	0.046	**25**	0.002	-0.132	**26**	-0.079	-0.013	**27**	-0.013	0.143
28	0	0.092	**29**	-0.013	0.143	**30**	-0.079	-0.013	**31**	0.002	-0.132
32	0.046	0.046	**33**	-0.132	0.002	**34**	-0.013	-0.079	**35**	0.143	-0.013
36	0.092	0	**37**	0.143	-0.013	**38**	-0.013	-0.079	**39**	-0.132	0.002
40	0.046	0.046	**41**	0.002	-0.132	**42**	-0.079	-0.013	**43**	-0.013	0.143
44	0	0.092	**45**	-0.013	0.143	**46**	-0.079	-0.013	**47**	0.002	-0.132
48	0.046	0.046	**49**	-0.132	0.002	**50**	-0.013	-0.079	**51**	0.143	-0.013
52	0.092	0	**53**	0.143	-0.013	**54**	-0.013	-0.079	**55**	-0.132	0.002
56	0.046	0.046	**57**	0.002	-0.132	**58**	-0.079	-0.013	**59**	-0.013	0.143
60	0	0.092	**61**	-0.013	0.143	**62**	-0.079	-0.013	**63**	0.002	-0.132

Tabla 3.2: *IFFT de los símbolos de la secuencia corta de entrenamiento*

Representamos los datos de la salida del IFFT de la tabla 3.2 en la figura 3.15 y vemos como la secuencia obtenida es periódica cada 16 muestras. Esto se debe a e que sólo las líneas espectrales con índices del -26 al 26 son múltiplos de 4 que no tienen amplitud cero, por lo que resulta una periodicidad de $T_{FFT}/4 = 0.8$ µs.

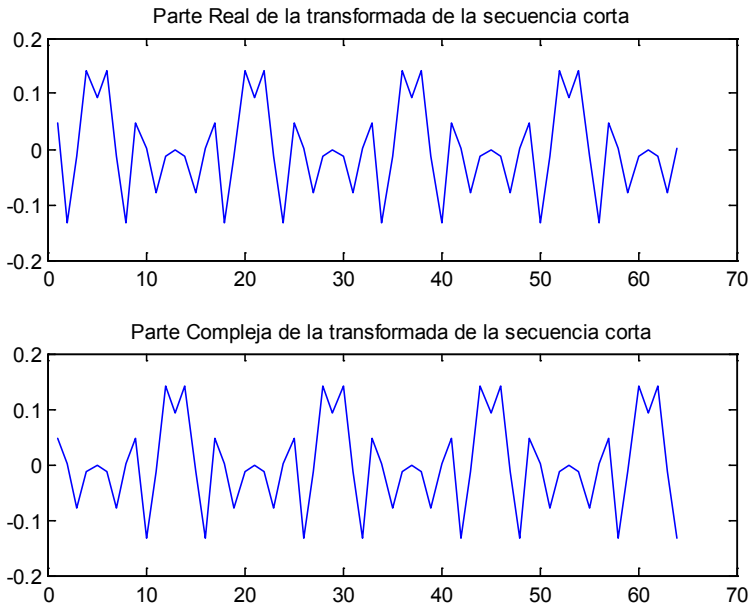


Figura 3.15: *IFFT de los símbolos de la secuencia corta de entrenamiento*

La secuencia corta se compone de diez veces este periodo de 0.8 µs (es decir, en total, 8 µs), lo que supone un total de 160 muestras.En las especificaciones del estándar 802.11a se añade una muestra de valor cero al final, con lo que en total tenemos una secuencia de 161 valores (figura 3.16).

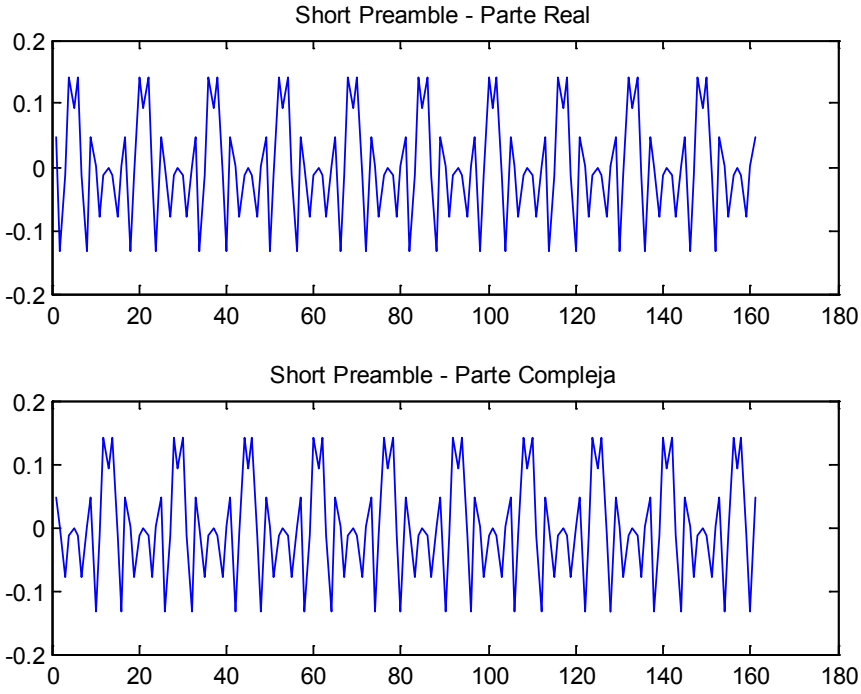


Figura 3.16: *Short Preamble*

3.2.5.2 Sincronización Temporal

Como ya se ha comentado en el punto anterior, la secuencia corta de entrenamiento busca sincronizar la recepción en el demodulador detectando el momento de inicio del paquete recibido.

Para calcular el inicio, se utiliza un algoritmo basado en una auto-correlación de la secuencia corta. Si la primera parte del símbolo de entrenamiento es idéntica en tiempo a la segunda parte excepto por la diferencia de fase producida por el offset de la frecuencia de portadora, entonces podemos determinar la primera muestra del paquete y también una buena estimación del offset de la frecuencia usando la siguiente métrica:

$$M(n) = \frac{|Q(n)|^2}{(R(n))^2} \tag{3.11}$$

Donde n es el índice de tiempo. El numerador se evalúa como:

$$Q(n) = \sum_{m=0}^{L-1} (r_{n+m}^* \cdot r_{n+m+L}) \tag{3.12}$$

Donde r_n es la muestra en el dominio del tiempo recibida, * es el complejo conjugado y *L* es el número de muestras en la primera parte del símbolo.

Por el otro lado, el denominador representa la energía recibida de la segunda parte del símbolo y se calcula usando:

$$R(n) = \sum_{m=0}^{L-1} |r_{n+m+L}|^2 \tag{3.13}$$

Características de la métrica y valores de L

L puede fijarse a 16 o a 32 muestras dependiendo del enventanado que se haga para el cálculo de la correlación. Si L=16, la secuencia corta de entrenamiento se divide en 5 símbolos diferentes. Sin embargo, una ventana mayor obtiene una estimación más estable debido a que en el cálculo de la correlación se usan más muestras. Un valor mayor de L, L=32 puede ser usado ya que la STS es lo suficientemente larga como para poder manejarlo. Hay que tener en cuenta que en ésta situación la STS estaría sólo dividida en dos símbolos diferentes.

Con el fin de mantener una ventana de 64 muestras (L=32) la tiene que ser estable usando sólo las primeras 96 muestras de la STS, es decir, los seis primeros símbolos. Según la implementación en [2] se puede de trabajar bien con los dos valores de L, aunque se han obtenido resultados mejores en las simulaciones cuando se eligió L=16.

Debido a las características de la correlación de la STS, la apariencia de la métrica es como vemos en la figura 3.16 [2].

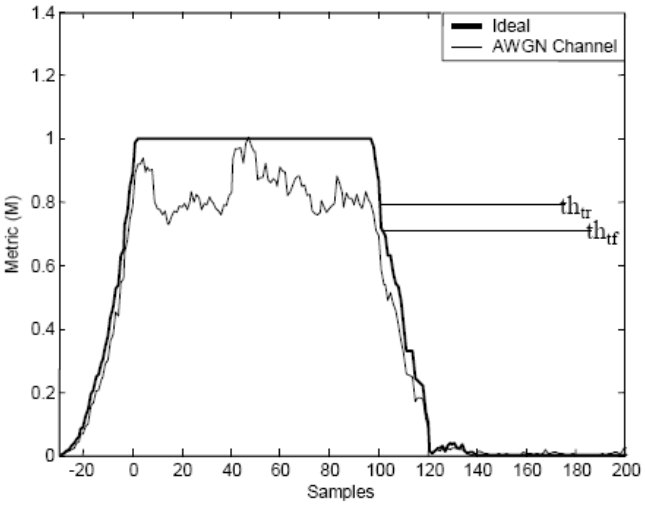


Figura 3.17: *Sincronización temporal. Valor de Mn*

La métrica puede ser interpretada como una auto-correlación normalizada de la secuencia de entrenamiento, con el objetivo de evitar la variación en potencia causada por las desviaciones del canal. Idealmente, cuando la métrica alcanza su valor máximo, se obtiene la sincronización.

La figura 3.17 muestra la métrica en una situación ideal (sin ruido o efecto del canal) y con ruido aditivo gaussiano AWGN cuando un cierto número de muestras de ruido (en este caso 30) se han insertado al principio del paquete en la simulación.

Vemos como la sincronización en el caso ideal es muy simple, y consiste en simplemente esperar a que la métrica valga 1. Mientras que para un escenario con AWGN es ligeramente diferente ya habrá que esperar a que la métrica alcance un umbral inferior.

Por tanto, el valor de este umbral se convierte en una cuestión importante, y el buen comportamiento del bloque de sincronización dependerá de él. Si el valor es muy bajo, la probabilidad de falsa alarma o de una temprana sincronización, crecerá. El valor óptimo del umbral se ha estudiado [2] mediante simulaciones, llegando a la conclusión de que depende del canal. Por ejemplo, para un canal AWGN, el valor está en torno a 0.6 y en canales multi-path en torno a 0.8.

4 DISEÑO DE UN SISTEMA DE COMUNICACIONES OFDM

4.1 INTRODUCCIÓN

En este capítulo llevamos a cabo el diseño de todos los bloques y sistemas necesarios que conforman un sistema de comunicaciones OFDM, implementándolo en el lenguaje VHDL. Simularemos a su vez las implementaciones tanto de cada bloque como del sistema completo para comprobar su correcto funcionamiento.

Al final, el objetivo es obtener un bloque modulador y otro demodulador que sean sintetizables en cualquier FPGA, dejando el estudio concreto de cómo programar el dispositivo SFF SDR de Lyrtech para el capítulo 5.

El capítulo se divide en dos bloques principales. En el primero (4.2), estudiamos los bloques principales de los que se componen los sistemas OFDM. Y en el segundo (4.3) integramos estos módulos para obtener un modulador y un demodulador OFDM.

4.2 COMPONENTES PRINCIPALES

4.2.1TRANSFORMADA DE FOURIER, DFT Y FFT

4.2.1.1 Introducción

Como vimos en el punto 3.2, para modular los símbolos de la constelación, en OFDM utilizamos la IFFT, y para demodularlos, la FFT. En este capítulo profundizaremos en la transformada de Fourier y en sus posibles implementaciones hardware.

La transformada de Fourier es una operación matemática en el tiempo continuo que nos transforma la información de una señal al dominio de la frecuencia. Revisamos las cuatro fórmulas de la transformada de Fourier:

- **Transformada de Fourier en tiempo continuo**

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)\cdot e^{j\omega t}\cdot d\omega \qquad (4.1)$$

- **Transformada de Fourier en frecuencia continua**

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)\cdot e^{-j\omega t}\cdot dt \qquad (4.2)$$

- **Transformada inversa de Fourier en tiempo discreto**

$$x[n] = \frac{1}{2\pi}\int_{0}^{2\pi} X(\Omega)\cdot e^{j\Omega n}\cdot d\Omega \qquad (4.3)$$

- **Transformada inversa de Fourier en frecuencia discreta**

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]\cdot e^{-j\Omega n} \qquad (4.4)$$

Al tratarse de tiempo continuo, para llevar a cabo su implementación en los diferentes sistemas digitales, se utiliza la transformada discreta de Fourier (DFT) (3.6) que no es más que la consecuencia de muestrear la ecuación 4.3.

Una forma más eficiente de calcular la DFT es la FFT (Fast Fourier Transform), ya que utiliza un número menor de operaciones. El algoritmo de FFT descompone la DFT de N puntos en transformadas más pequeñas.

Para implementar la FFT existen dos procedimientos: diezmado en el tiempo (DIT) y diezmado en frecuencia (DIF). En nuestro caso, emplearemos el primer tipo, como veremos posteriormente.

4.2.1.2 Algoritmos FFT

A) Introducción

Los algoritmos FFT/IFFT [28] [29] se realizan por etapas; la realización de una etapa supone el cómputo de operaciones básicas sobre los elementos de la entrada para calcular los elementos de la salida. Estas operaciones básicas se conocen con el nombre de *butterflies* (mariposas) para el caso de los algoritmos llamados Radix-2 y con el nombre de *dragonfly* cuando los algoritmos son Radix-4.

Una *butterfly* o mariposa supone el cálculo de dos elementos de la salida de la etapa a partir de únicamente dos elementos de la entrada de la etapa. Una *dragonfly* calcula simultáneamente cuatro elementos de la salida de la etapa a partir de únicamente cuatro elementos de la entrada de la etapa en cuestión. Los algoritmos Radix-2 precisan $\log_2 N$ etapas y los Radix-4 $\log_4 N$, siendo N el tamaño de los vectores de entrada. En la práctica no suelen utilizarse algoritmos Radix-8 o superiores para el cómputo hardware de la IFFT/FFT.

En nuestro caso, hemos llevado a cabo la implementación mediante el uso de "IP Cores" de un módulo FFT Radix-4. La motivación viene dada por la comparativa entre ambos métodos. Mientras los algoritmos Radix-2 emplean mínimos recursos, los algoritmos Radix-4 son más eficientes, ya que, por lo general, la FFT Radix-4 requiere sólo el 75% de las multiplicaciones complejas que Radix-2 FFT.

Podemos ver este aspecto en la tabla 4.1, que compara el coste computacional de ambos algoritmos:

	Multiplicaciones reales				Sumas Reales			
N	**Radix-2**	**Radix-4**	**Radix-8**	**Split Radix**	**Radix-2**	**Radix-4**	**Radix-8**	**Split Radix**
16	24	20		20	152	148		148
32	88			68	408			388
64	264	208	204	196	1032	976	972	964
128	72			516	2054			2308
256	1800	1392		1284	5896	5488		5380
512	4360		3204	3076	13566		12420	12292
1024	10248	7856		7172	30728	28336		27652

Tabla 4.1: *Coste computacional algoritmos FFT*

En el siguiente punto pasamos a explicar con más detalle el funcionamiento del algoritmo FFT, Radix-4.

B) Radix-4

La FFT Radix-4 lleva a cabo particiones recursivas de una FFT en cuatro subgrupos a los que aplicar FFTs menores. Las salidas de estas FFTs son reutilizadas para calcular de nuevo algunas salidas, reduciendo el coste computacional. Para poder llevarse a cabo, la longitud de la DFT ha de ser múltiplo de 4.

El algoritmo Radix-4 separa la secuencia original en 4 subsecuencias:

$$x(4n), x(4n+1), x(4n+2), x(4n+3), \quad n = 0,1,\dots, {}^{N}/_{4} - 1 \qquad (4.5)$$

$$X(p,q) = \sum_{l=0}^{3} \left[W_n^{l\,q} F(l,q) \right] W_4^{l\,p} \qquad (4.6)$$

$$F(l,q) = \sum_{m=0}^{({}^{N}/_{4})-1} x(l,m) W_{N/4}^{m\,q} \qquad (4.7)$$

$$F(l,q) = \sum_{m=0}^{({}^{N}/_{4})-1} x(l,m) W_{N/4}^{m\,q} \qquad (4.7)$$

$$p = 0,1,2,3; \quad l = 0,1,2,3; \quad q = 0,1,2,\dots,\frac{N}{4} - 1 \qquad (4.8)$$

Donde el factor W se conoce “twiddle-factor” y viene dado de la separación de la DFT en cuatro subgrupos.

$$X(k) = \sum_{n=0}^{N-1} x[n] W^{nk} \quad k = 0,1,\dots,N-1 \qquad (4.9)$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W^{-nk} \quad n = 0,1,\dots,N-1 \qquad (4.10)$$

donde $W = e^{-j2\pi/N}$ (4.11)

Lo que nos da el cálculo de un punto de la transformada de Fourier de la siguiente forma:

$$X(k) = x[0]W^0 + x[1]W^k + x[2]W^{2k} + \cdots + x[N-1]W^{k(N-1)}, k = 0,1,\dots,N-1 \qquad (4.12)$$

Que es la ecuación de transformada de Fourier en el punto *k* de Radix-4

Siguiendo esta ecuación, las cuatro DFTs (*F(l,q)*) obtenidas de las ecuaciones, son combinadas para obtener la DFT de longitud *N*. La expresión para combinar las *N/4* DFTs define un dragonfly diezmado-en-tiempo (*decimation-in-time dragonfly*), que se llama así debido a que las muestras de tiempo son reordenadas en grupos alternos.

Dicha ecuación puede ser expresada en forma de matriz de la siguiente forma:

$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F(0,q) \\ W_N^q F(1,q) \\ W_N^{2q} F(2,q) \\ W_N^{3q} F(3,q) \end{bmatrix} \tag{4.13}$$

Estas operaciones, dibujadas quedan de la siguiente manera. Notar que cada dragonfly envuelve tres multiplicaciones complejas desde W_N^0 = 1, y 12 sumas complejas.

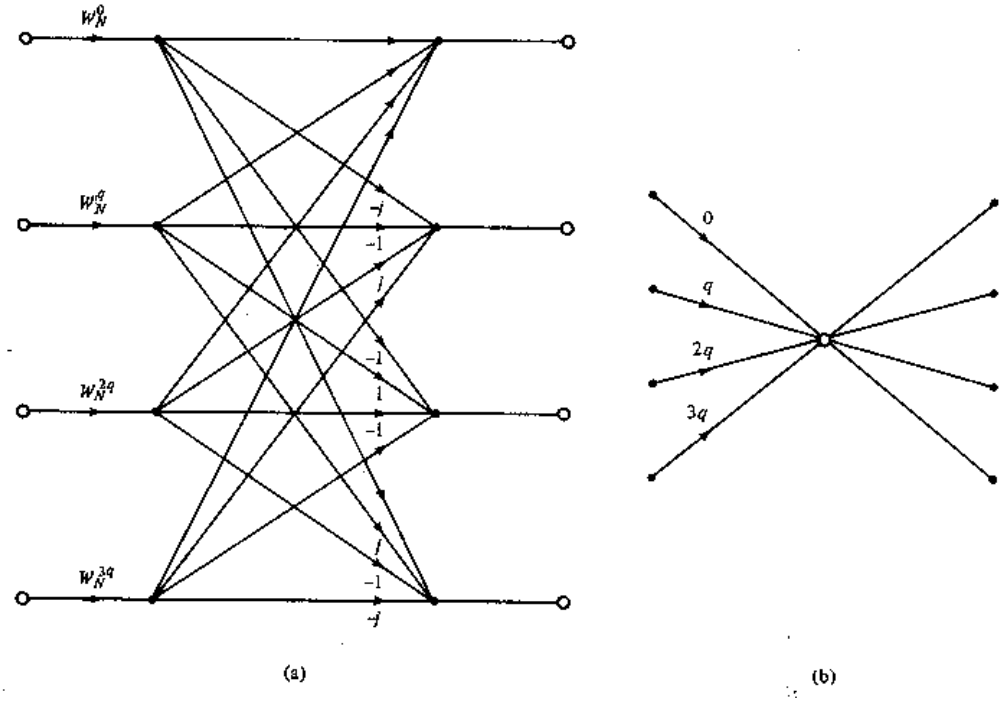


Figura 4.1: *Diagrama del algoritmo dragonfly diezmado en tiempo de Radix-4*

Para la mejorar la eficiencia de este algoritmo, se suelen reducir las sumas complejas de los dragonfly de 12 a 8 si empleamos un producto de matrices como el siguiente:

$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & -j & 0 & j \\ 1 & 0 & 1 & 0 \\ 1 & j & 0 & -j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 \\ 0 & j & 0 & -j \end{bmatrix} \begin{bmatrix} W_N^0 F(0,q) \\ W_N^1 F(1,q) \\ W_N^2 F(2,q) \\ W_N^3 F(3,q) \end{bmatrix} \tag{4.14}$$

Lo cual se traduce en el siguiente esquema

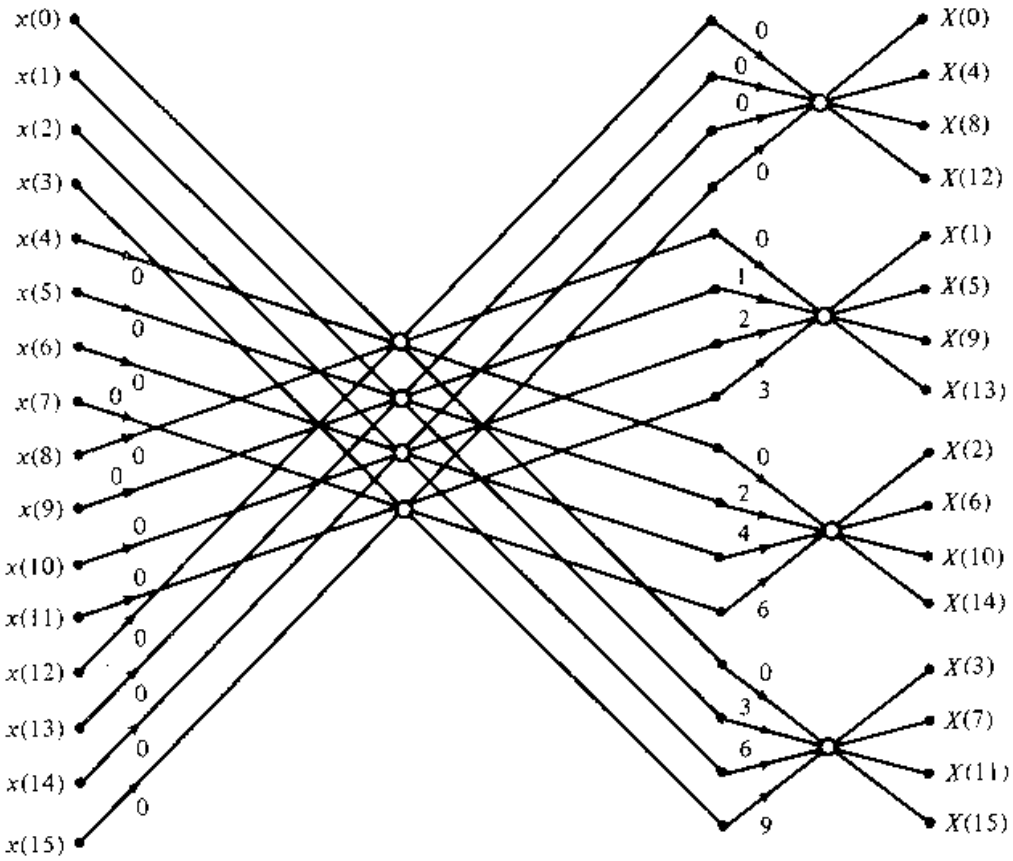


Figura 4.2: *FFT-Radix 4 de 16 puntos. Algoritmo diezmado-en-tiempo con entrada en orden y salida en orden inverso.*

Debido a la periodicidad *N/4* de las DFTs, la muestra de la frecuencia *k*, puede ser reutilizada para el cálculo de *X(k), X(k+N/4), X(k+N/2), X(k+3N/2).* Esta propiedad es la que da al algoritmo Radix-4 su eficiencia. La computación involucrada en cada grupo de muestras de cuatro frecuencias, constituye una Radix-4 dragonfly, como se muestra en la figura 4.3.

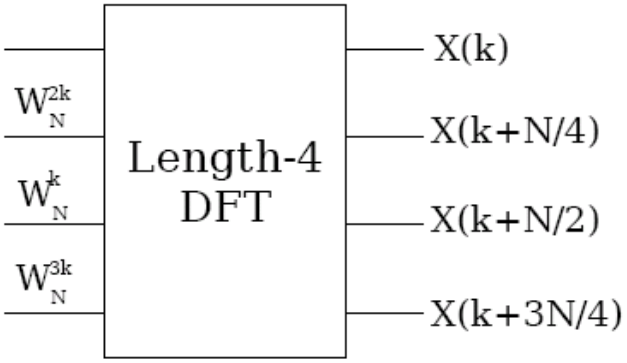


Figura 4.3: *Esquema de un bloque FFT-Radix 4*

4.2.1.3 FFT Core de Xilinx

Para llevar a cabo la implementación en la FPGA del algoritmo FFT explicado, nos hemos apoyado en el los módulos “IP Cores” dados, que vienen facilitados en el software de Xilinx. En nuestro caso, contamos con la licencia del Xilinx 9.2i que nos permite el uso del core FFT v3.1 [30].

Como ya hemos comentado en el punto anterior, para la implementación del bloque FFT hemos elegido un algoritmo Radix-4. En la documentación del core, podemos ver el siguiente esquema de cómo está llevado a cabo dicho algoritmo.

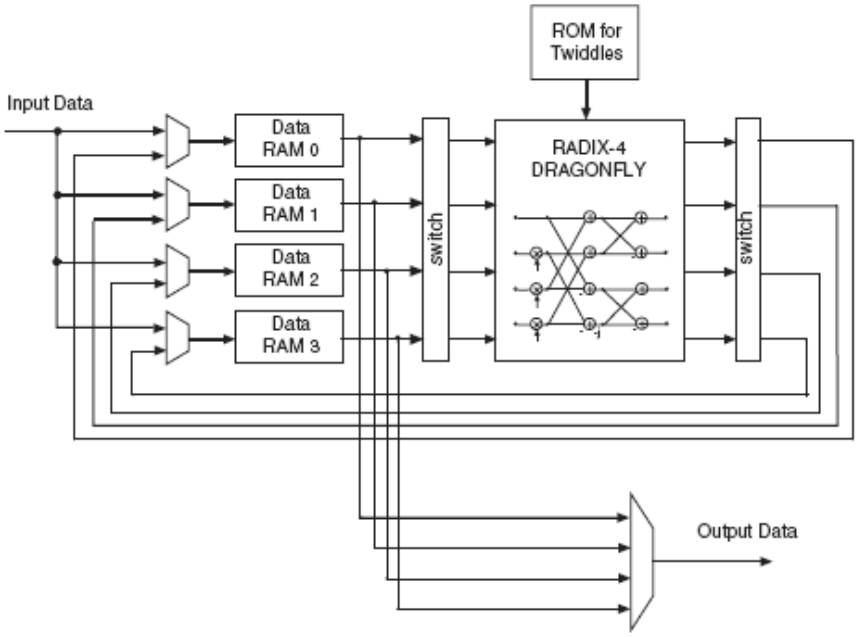


Figura 4.4: *Esquema la FFT-Radix 4 del core de Xilinx*

En el esquema de la figura 4.4 [30] podemos encontrar los elementos explicados en el punto anterior, la división en cuatro secuencias y el diezmado en tiempo y el algoritmo dragonfly.

Accediendo al resto de la documentación, [30] y [31], podemos ver las especificaciones del core con las que llevamos a cabo el diseñado del módulo, la máquina de estados y las simulaciones para obtener conclusiones sobre el funcionamiento del mismo.

A) Criterios de diseño

El *core* FFT tiene la siguiente definición de entradas y salidas, donde distinguimos entre las señales de control y los datos, de los cuales pasamos a describir brevemente los más importantes.

- **Datos:**

- *XN_RE, XN_IM* son las señales de entrada y *XK_RE, XK_IM* de salida

- *XN_INDEX, XK_INDEX* indican los índices de los datos a la entrada y salida respectivamente.

- *CLK* es el reloj

- *START*: señal de comienzo

- *UNLOAD*: descarga de resultados

- *NFFT*: tamaño de la FFT

- *FWD_INV*: indica FFT/IFFT

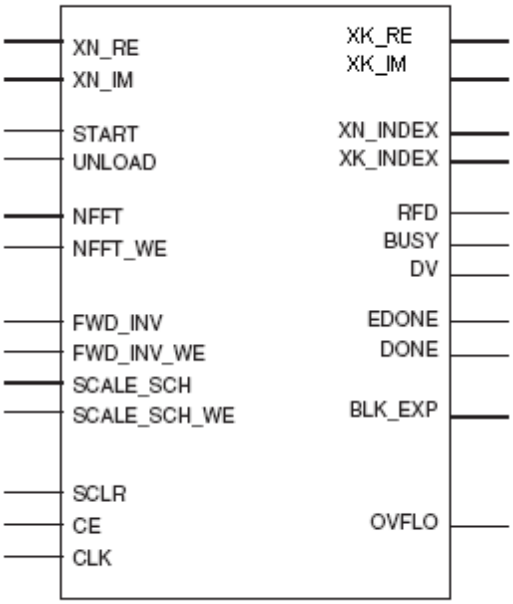


Figura 4.5: *Bloque FFT V3.1*

De las opciones que nos da el módulo FFT del core de Xilinx, se ha implementado con las siguientes características:

- Algoritmo Radix-4
- Tamaño de datos de entrada: 16bits
- Tamaño de puntos de la FFT, variable desde 64 puntos hasta 1024.
- Redondeo de datos de salida por truncamiento
- Almacenamiento en memorias RAM bloque

B) Máquina de estados

Para la integración del bloque FFT en el resto del sistema, hemos construido un bloque que mediante una máquina de estados, gobierne la FFT. Las señales de control de la FFT las habilitamos en nuestra máquina de estados siguiendo las pautas que sugieren las hojas características. Así, observando el cronograma de la figura 4.6 obtenemos las siguientes conclusiones:

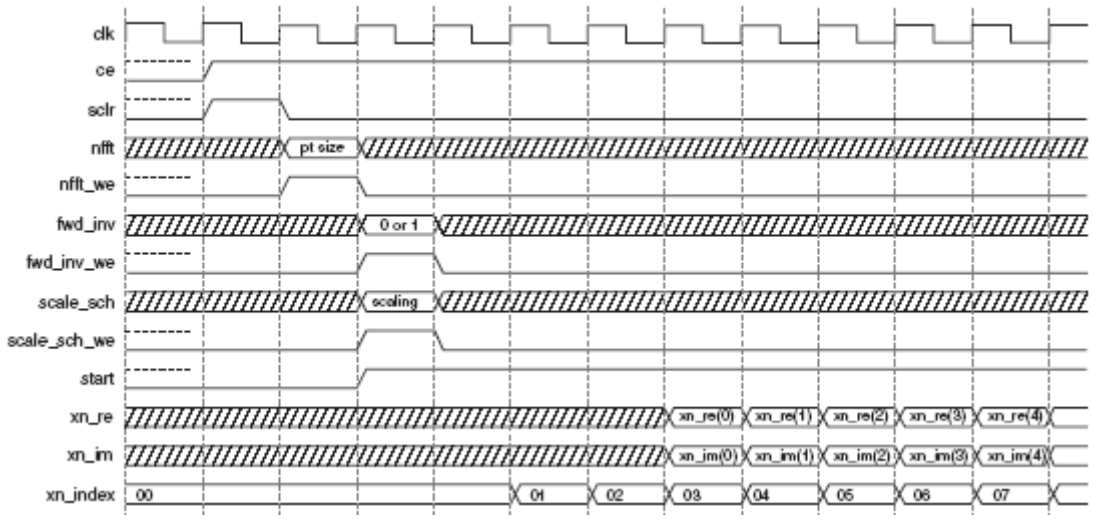


Figura 4.6: *Cronograma FFT V3.1*

- El orden de la FFT para que sea válido, la señal *nfft* tiene que permanecer constante y mientras que su *enable*, *nfft_we*, está a nivel alto durante un ciclo de reloj.

- La transformada directa hay que activarla con la señal *fwd_inv* a nivel alto. Esta señal tiene que permanecer estable mientras su *enable*, *fwd_inv_we*, está activo a nivel alto durante un ciclo de reloj.

- Las señales de control *start* y *ce* son activas a nivel alto.

Con esta información, llevamos a cabo dos módulos: *modulo_ifft* y *modulo_fft* para ambas partes del sistema de comunicaciones, modulador y demodulador. El diseño del bloque de ambos módulos es el mismo (figura 4.7) e implementarán una máquina de estados que configure el *core fft* con la única distinción de utilizar las señales *ifft_fwd_inv* y *ifft_fwd_inv_we* de forma que se escoja el modo de operación correspondiente (FFT o IFFT).

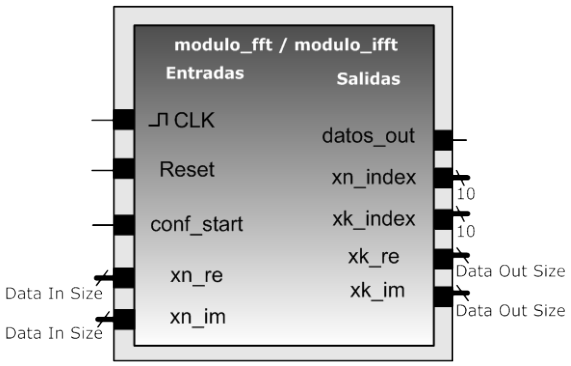


Figura 4.7: *Bloque modulo_fft / modulo_ifft*

Vemos la máquina de estados de ambas configuraciones:

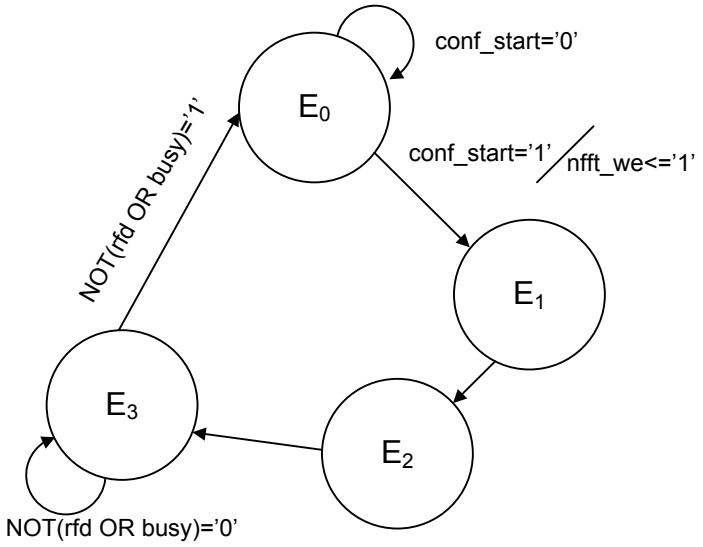


Figura 4.8: *Máquina de estados de moluloFFT/moduloIFFT*

En cada uno de los estados, se configuran las señales del siguiente ciclo de reloj según la tabla 4.2. El proceso del paso por los estados de los módulos de configuración del core fft, es directo, y simplemente requerirá la activación de la señal de comienzo conf_start. El control del módulo no dejará una nueva inicialización hasta que no dé por finalizado el cálculo de la FFT/IFFT, para asegurarlo comprobará las señales busy y rfd.

Estado	E0	E1	E2	E3	Todos
Señales	nfft_we<=0 start<=0 fwd_inv<=0 fwd_inv_we<=0	nfft_we<=0 start<=1 fwd_inv<=1/0 fwd_inv_we<=1/0	nfft_we<=0 start<=0 fwd_inv<=0 fwd_inv_we<=0	nfft_we<=0 start<=1 fwd_inv<=0 fwd_inv_we<=0	nfft<=SeleccionaN unload<edone

Tabla 4.2: *Señales de los estados. Máquina de estados de moluloFFT/moduloIFFT*

Para la descarga de los datos, simplemente desde el modulador, esperaremos la señal de que los datos ya se han calculado para activar la señal *unload* y que el módulo ponga los datos a la salida.

Vemos en la figura 4.9 sacada de la documentación del core [30], un posible cronograma de la descarga de datos, en donde, desde que el módulo recibe la señal de *unload*, hasta que comienza a poner los datos a la salida, pasan varios ciclos de reloj que tendremos en cuenta a la hora de implementar el modulador completo.

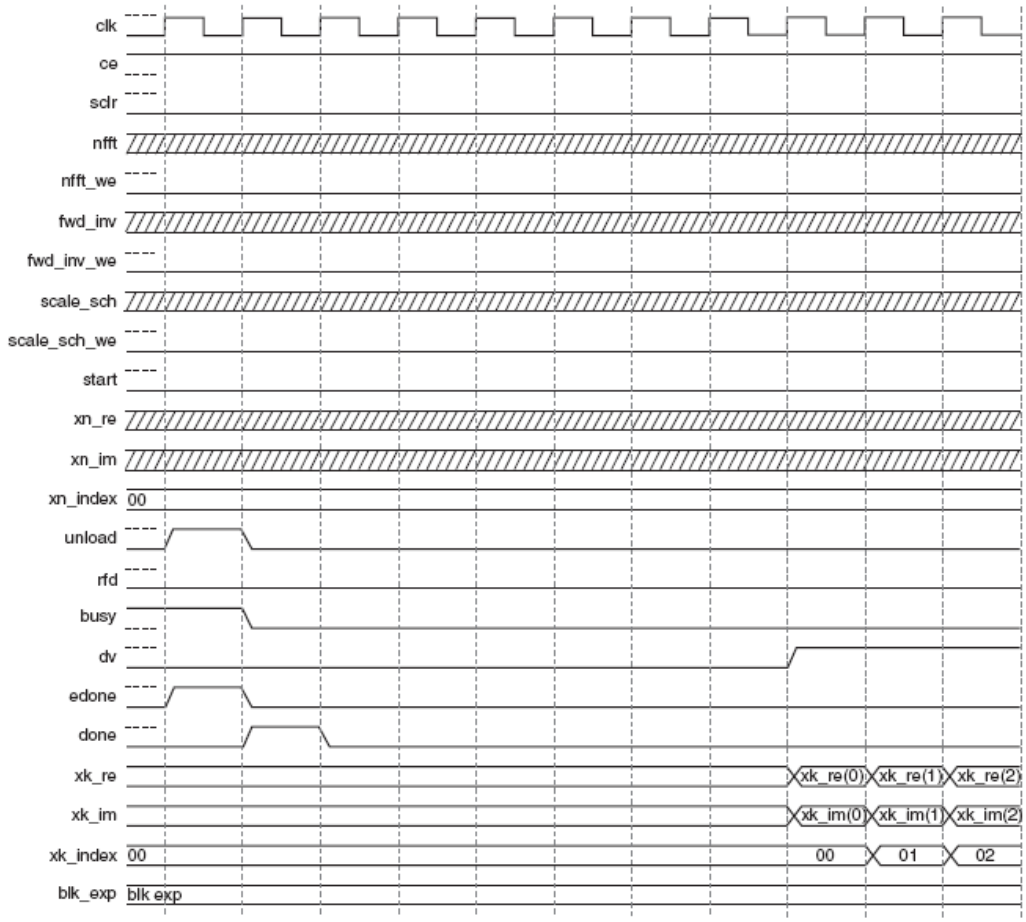


Figura 4.9: *Cronograma de descarga de datos del core FFT v3.1*

C) Resultado de las simulaciones

En este punto, pasamos a representar los resultados de las simulaciones de la implementación llevada a cabo. En el primer gráfico (4.10) podemos observar cómo se configura el *core* FFT, mediante la máquina de estados de la figura 4.8, para que comience la carga de datos.

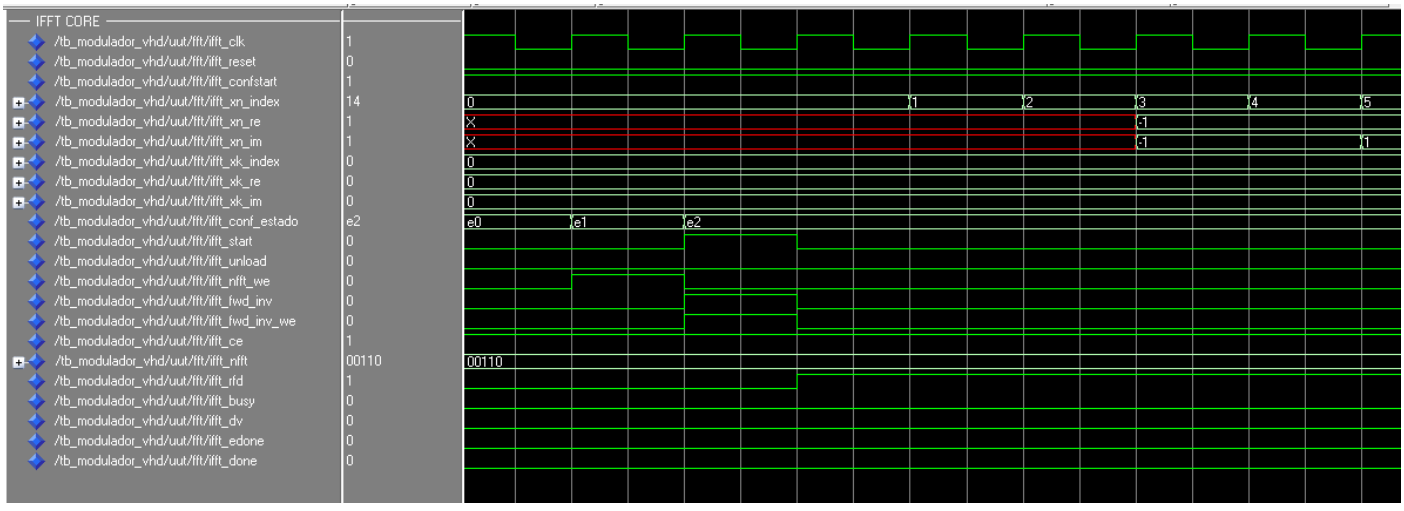


Figura 4.10: *Simulación FFT. Cronograma 1: Comienzo de configuración*

El proceso continua hasta que alcanzamos el tamaño de puntos de la FFT, N. En nuestro caso, la FFT es de 64 puntos, por lo que el último dato que recibe, lo hace en el índice 63. Al acabar, la señal *busy* se pone a nivel alto, y comienza el cálculo de la FFT.

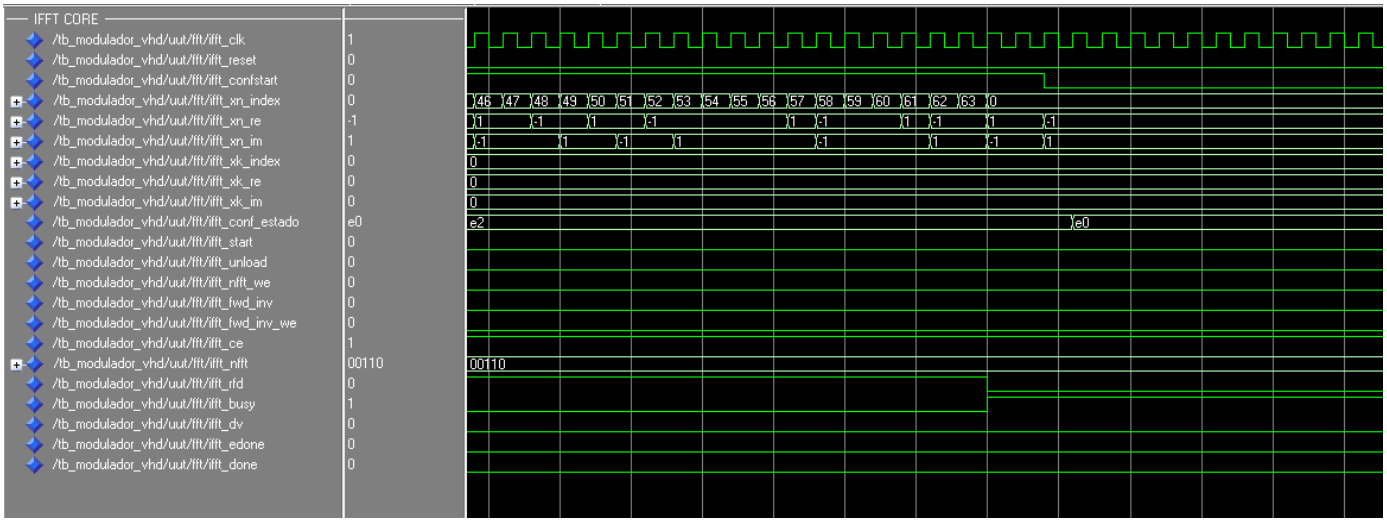


Figura 4.11: *Simulación FFT. Cronograma 2: Fin de carga de datos*

Cuando el algoritmo termina de calcular, nos lo indica mediante las señales *busy*, *done* y *edone*. Y siguiendo el cronograma de la documentación, ponemos la señal de *unload* a nivel alto a la vez que *edone*, y ocho ciclos de reloj después, comienza a sacar los valores de la FFT calculada, indicándolo mediante la señal *dv*.

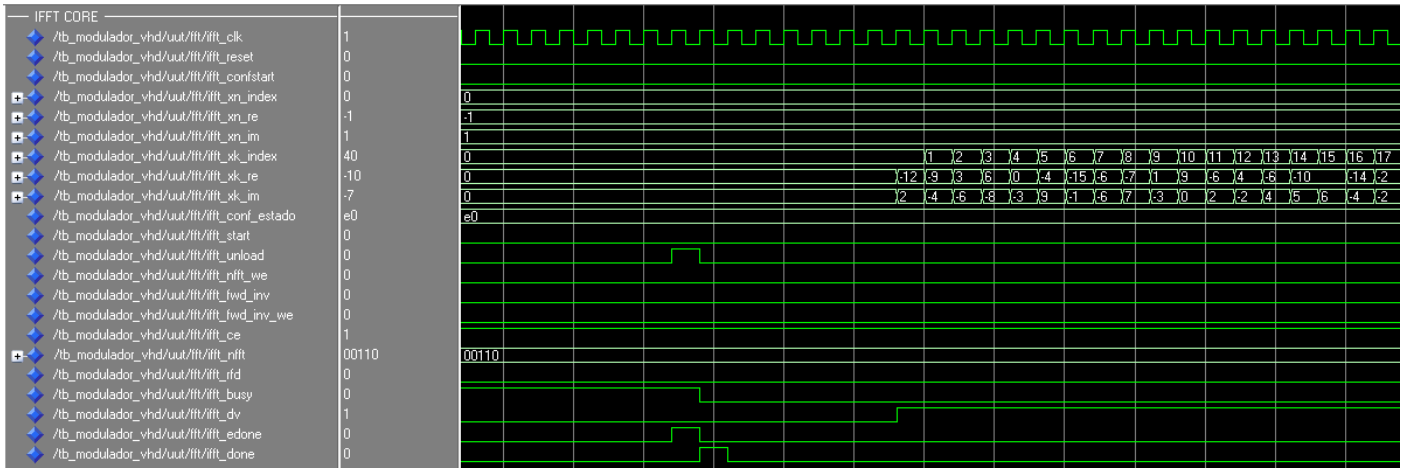


Figura 4.12: *Simulación FFT. Cronograma 3: Descarga de datos*

4.2.2 CONVERSOR SERIE/PARALELO Y ASOCIACIÓN DE SÍMBOLOS

4.2.2.1 Introducción

En este apartado pasamos a estudiar y diseñar los bloques del conversor serie-paralelo y del asociador de símbolos. Mirando estos dos bloques como un conjunto, el objetivo es conseguir una transformación de los bits serie que van llegando al modulador, en N símbolos paralelos que introducir en la FFT.

4.2.2.2 Diseño

Para llevar a cabo la implementación de ambos bloques, primero hemos de tener en cuenta la diferencia entre los circuitos combinacionales y los secuenciales. Un sistema combinacional es aquel en que las salidas en un instante sólo dependen de las entradas en aquel instante. En cambio, un automatismo secuencial es aquel en el que las salidas en cada instante no dependen sólo de las entradas en aquel instante sino que también dependen de los estados anteriores y de su evolución.

Para el diseño de los bloques del conversor serie-paralelo y del asociador de símbolos, hemos juntado ambos en un solo módulo debido a que no la carga de los símbolos de la constelación en el módulo IFFT se hace de forma serie. Así, cuando el conversor serie-paralelo tenga el tamaño de bits de una palabra de la constelación, el modulo es capaz de poner a la salida los símbolos (real e imaginario) correspondientes directamente.

El diseño del módulo “bit2simb” es el siguiente:

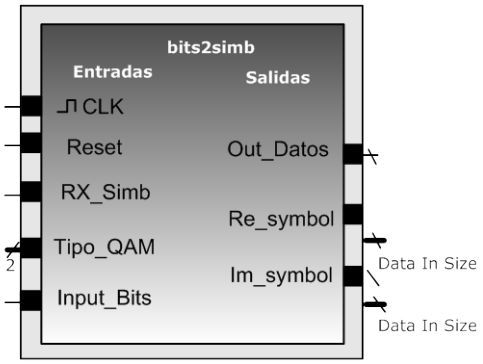


Figura 4.13: *Bloque bit2simb*

- **Señales de entrada:**

  - *CLK*: reloj del módulo
  - *Reset*: activa a nivel alto
  - *Rx_bits*: indica al módulo que se están recibiendo bits
  - *Tipo_QAM*: 4, 8, 16 ó 64 QAM
  - *Input_bits*: bits de entrada

- **Señales de salida:**

    - *Out_Datos*: indica que se están poniendo datos a la salida
    - *Re_symbol*: símbolo real
    - Im_symbol: símbolo imaginario

La asociación de la constelación, se lleva a cabo empleando código Gray, de forma que la distancia entre dos símbolos contiguos sea de un bit. El diseño del modulo lo llevamos a cabo utilizando la constelación 4-QAM, pero dejando el bloque diseñado para el futuro uso para las constelaciones 8-QAM, 16-QAM y 64-QAM.

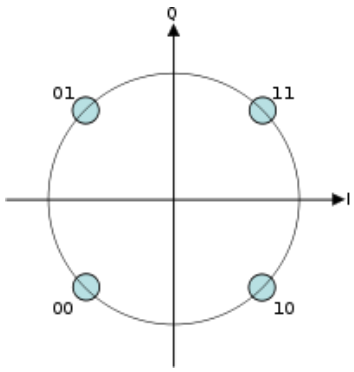


Figura 4.14: *Constelación 4-QAM*

4.2.2.3 Funcionamiento y simulación

El funcionamiento del módulo es el siguiente. Cuando se activa la señal de *Rx_bits*, se empieza a almacenar cada el valor del bit de entrada en un registro del tamaño de la palabra de la constelación, es decir, 2 para 4-QAM.

Cuando se ha completado una palabra completa, se hace la asociación de símbolo, y se pone a la salida el valor real e imaginario del símbolo. Al mismo tiempo se indica poniendo *out_datos* a nivel alto.

Comprobamos su correcto funcionamiento del módulo construyendo un banco de pruebas que vaya introduciendo bits en la entrada, y comprobamos los símbolos de salida en el cronograma de la figura 4.15.

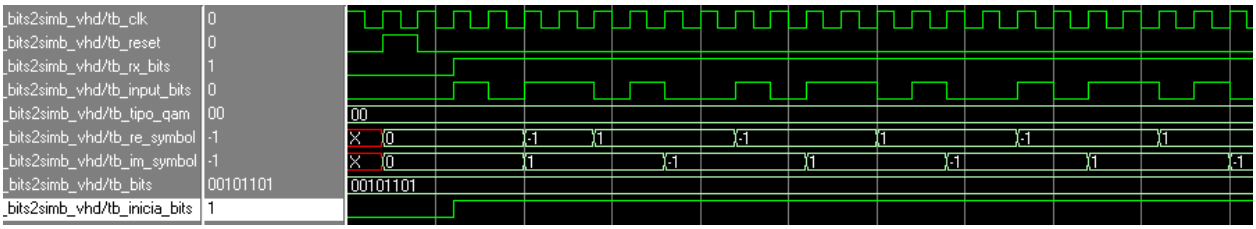


Figura 4.15: *Simulación Bit2Simb. Asociación de bits*

4.2.3 CONVERSOR PARALELO/SERIE Y RECUPERACIÓN DE BITS

4.2.3.1 Introducción

En este apartado vamos a ver el bloque contrario al visto en el punto anterior, es decir, pasamos a estudiar y diseñar los bloques del conversor paralelo-serie y recuperación de los bits de información a partir de los símbolos de la constelación recibidos.

4.2.3.2 Diseño

De la misma manera que se hizo en el diseño del bloque *simb2bits*, ambas operaciones se llevan a cabo dentro de un mismo bloque. En primer lugar nos referimos al bloque conversor serie a paralelo.

A) Serie-Paralelo

Hay que tener en cuenta que éste bloque ira en el demodulador, que tras hacer la transformada de Fourier sobre los símbolos OFDM recibidos, obtendrá a la salida los símbolos asociados a la constelación. Como el módulo dado en el entorno de desarrollo (*FFT Core*) va sacando cada dato en cada ciclo de reloj, obtenemos la salida en serie, con lo que las operaciones de reordenamiento del bloque paralelo-serie, se llevan a cabo internamente en el core FFT.

Aprovechando ésta circunstancia, podemos asociar de forma combinacional los bits de información que fueron enviados en función de los símbolos a las salida de la FFT. La justificación del uso de un bloque combinacional es que las operaciones de decisión y asociación de bits a la salida tienen un coste computacional bajo, por lo que el tiempo de cálculo será muy inferior al periodo de reloj.

B) Decisor

Para llevar a cabo la implementación del decisor de símbolo, hemos utilizado el comparador de las librerías IP del Xilinx. Dicho módulo, tiene el siguiente esquema:

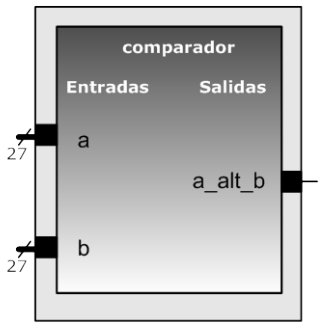
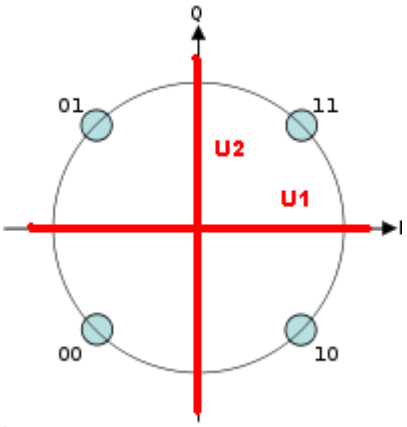


Figura 4.16: *Módulo comparador*

El comparador trabaja de forma combinacional, de forma que la señal *a_lt_b* está nivel alto cuando se cumple la relación a<b, y a nivel bajo cuando no se cumple.

Es decir, $$a\_lt\_b = \begin{cases} 1 & si \ \ a < b \\ 0 & si \ \ a \geq b \end{cases} \qquad (4.15)$$

Como ya comentamos en el punto 4.2.2, en la implementación completa, usaremos una constelación 4-QAM. Por tanto, en un decisor de una constelación 4-QAM con símbolos de la misma energía y equiprobables, los umbrales de decisión coinciden con los ejes de fase y cuadratura. Por lo que el decisor de símbolo se simplifica a comparar con cero, o lo que es lo mismo, a decidir en función del signo del símbolo. En la figura 4.17 observamos los umbrales de decisión sobre la constelación y en la tabla 4.3 la asociación símbolo-bits.



Signo Real	Signo Imaginario	Bits
+	+	11
+	-	01
-	+	10
-	-	00

***Figura 4.17**: Decisor 4-QAM*

***Tabla 4.3:** Asociación símbolos-bits*

Utilizando el comparador descrito con la señal *b* igual a cero, implementamos un proceso que nos asocie los símbolos según la señal *a_lt_b* para las entradas real e imaginaria. En la figura 4.18 vemos como queda el diseño de este bloque.

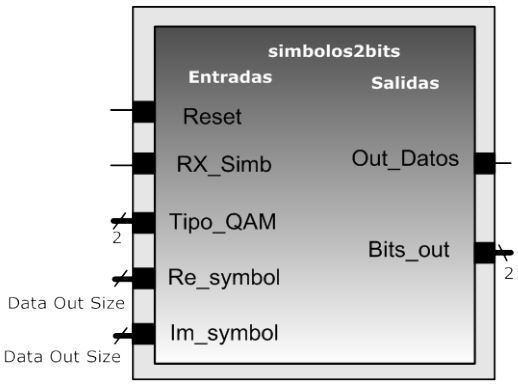


***Figura 4.18:** Bloque Decisor y Asociador de Bits comparador*

C) Funcionamiento y simulación

El diseño del bloque *simb2bits* es asíncrono ya que trabaja de forma combinacional. En el momento en que se pone a nivel alto la señal que informa de que está recibiendo símbolos *("s2b_Rx_simb"*) comienza a funcionar y a poner a la salida los bits calculados.

Vemos en la simulación (figura 4.19) el proceso que lleva a cabo el bloque y como coincide la asociación a la de la tabla 4.3 (Asociación símbolos-bits).

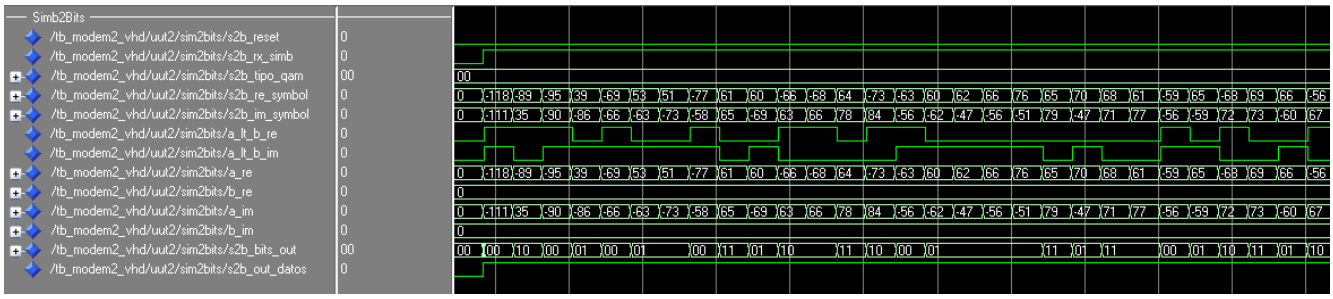


Figura 4.19: *Simulaciones del decisor y asociador de bits*

4.2.4 MEMORIAS RAM

4.2.4.1 Introducción

En este apartado pasamos a estudiar el funcionamiento y significado de las memorias en una implementación VHDL; así como las diferentes opciones que existen. Por último llevaremos a cabo a la implementación de una memoria RAM síncrona.

4.2.4.2 Memorias en VHDL

En los circuitos digitales, la unidad mínima de almacenamiento es un biestable o "Flip-Flop", ya que puede mantener un bit de información por un tiempo indefinido. El paso de un estado a otro de la información almacenada se realiza variando las entradas del biestable.

Uniendo cierto número de estos elementos se puede formar lo que se conoce como "*Registro"*, el cual se puede utilizar para guardar una palabra de n bits que representa un dato más complejo. Del mismo modo, uniendo Registros, se pueden almacenar datos más complejos como un "String*"* o un "array*"*. Por tanto, cualquier información almacenada en una implementación VHDL se puede reducir al valor de los estados de muchos biestables.

Normalmente, los módulos FPGA vienen provistos con memorias internas donde se almacenan los datos y el código del programa. Existen dos tipos de memorias incluidas en el chip de las FPGAs de Xilinx: memoria bloque y memoria distribuida.

La memoria RAM de bloque viene incluida en el chip de la FPGA y tiene un cierto número de bloques dedicados a ser utilizados como memoria RAM. Dentro de cada bloque, se puede configurar una tabla que apunte a otra memoria. Combinando varios bloques de esta manera para obtener una memoria mayor es lo que se conoce como memoria distribuida.

Mientras que el uso de memoria distribuida utiliza un mayor recurso del procesador, la memoria de bloque tiene un tamaño limitado. Por lo que ambas opciones pueden ser validas en función de la implementación llevada a cabo.

En nuestro caso, la elección del tipo de memoria de almacenamiento la llevará el programa en el momento de sintetizar, optimizando para la FPGA (Virtex-4) utilizada, como veremos en el punto 5.2.1.3.

4.2.4.3 Diseño

El diseño del bloque de memoria es muy simple, ya que no necesita ningún proceso de control. El bloque se compone de cinco entradas y de una sola salida. Como ya comentamos, se trata de una memoria síncrona, por lo que una de las entradas necesita que sea el reloj del sistema (*CLK*). Tiene una entrada de activación a nivel alto, *enable* y dos entradas para elegir el modo de operación: leer (*Read*) o escribir (*Write*).

Además de la dirección de memoria del dato que queremos leer o escribir. Hay dos posibilidades de tamaño para la entrada *address*, ya que en el sistema completo, usaremos dos tipos de memorias, una para almacenar los N símbolos de la constelación calculados para hacer la IFFT, y otro tipo donde almacenaremos los símbolos OFDM que se van calculando antes de transmitir. La única diferencia entre ambas, es el tamaño del número de datos que almacena.

Por último, tenemos la señal de entrada con el dato a almacenar, *Indata* y la señal de salida con el dato a leer *Outdata*.

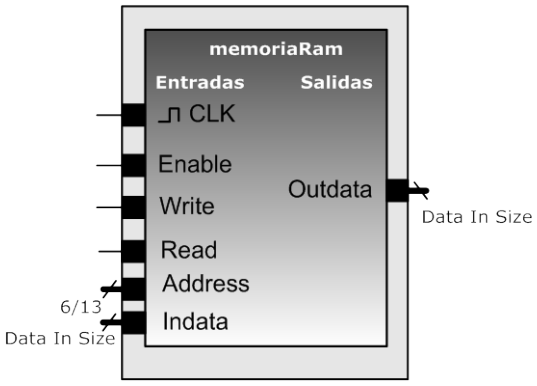


Figura 4.20: *Bloque memoriaRam*

4.2.4.4 Funcionamiento y simulación

El funcionamiento de la memoria RAM es el siguiente. Sólo funciona si la señal *enable* está activa. Para escribir datos en la memoria, se pone a nivel alto la señal *write*, y se pone en *address* la dirección de memoria en donde se quiere almacenar el dato *indata.* En el siguiente ciclo de reloj, el dato queda almacenado, y la señal de *write* vuelve a cero.

En la figura 4.21 podemos observar, como cada dos ciclos de reloj, llega un nuevo dato a almacenar en la memoria, en orden creciente.

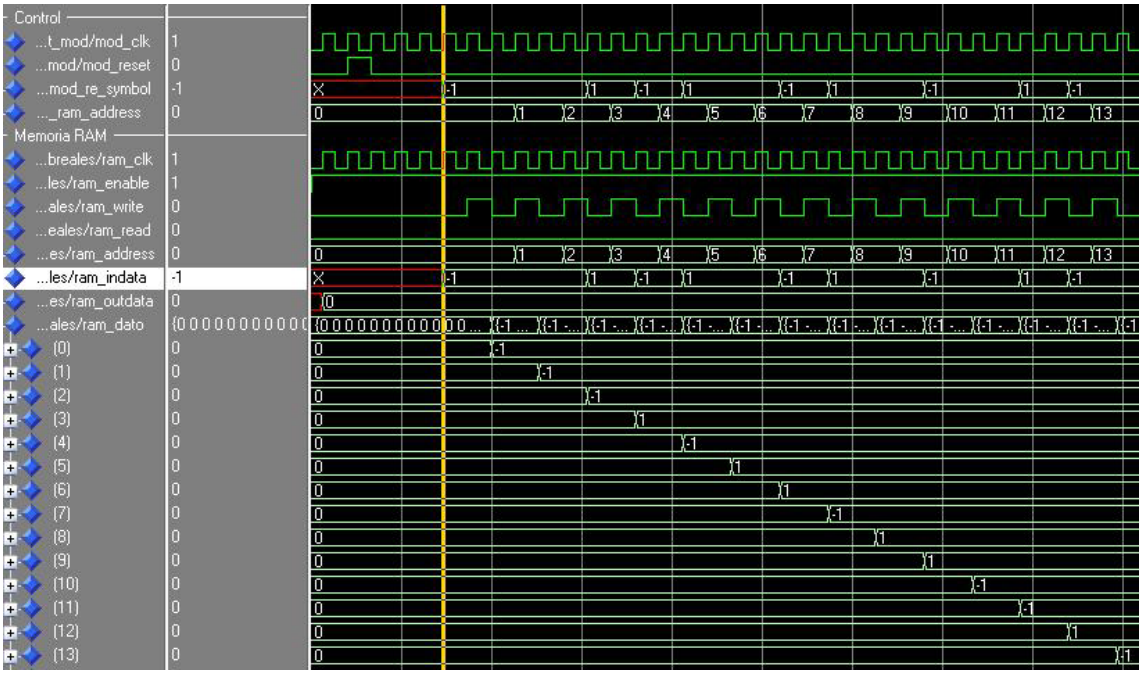


Figura 4.21: *Grabación de datos en la memoriaRam*

Para la lectura de datos, el proceso es similar. Mientras la señal *read*, no esté activa, el dato de salida estará a cero. Cuando se activa la señal de *read*, se pone a la salida en el siguiente ciclo de reloj el dato guardado en la dirección *address*.

En la figura 4.22 vemos como en este caso, la señal de *read* permanece activa, lo que supone que los datos leídos son puestos a la salida en cada ciclo de reloj.

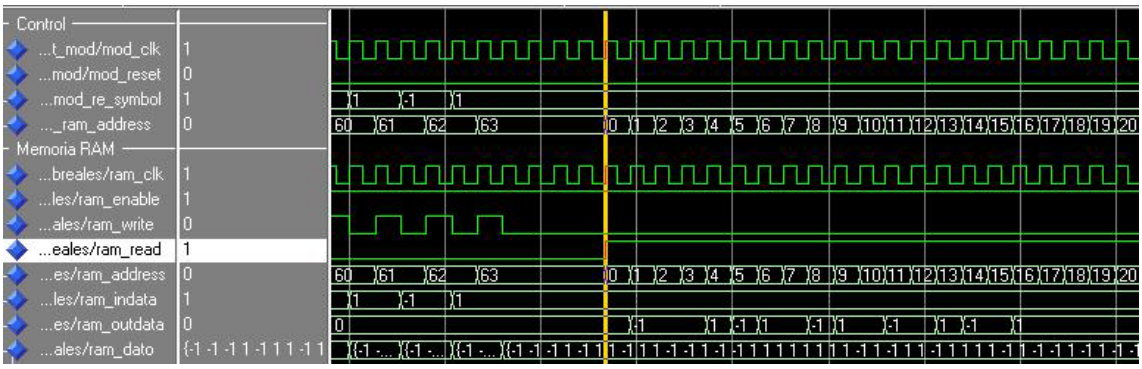


Figura 4.22: *Grabación de datos en la memoriaRam*

4.2.5 SINCRONIZACIÓN

4.2.5.1 Introducción

En el presente proyecto, se ha llevado a cabo una sincronización temporal utilizando para ello la secuencia corta de entrenamiento. En este apartado diseñamos y simulamos el funcionamiento de las secciones dedicadas a la sincronización tanto en el bloque del modulador como del demodulador.

4.2.5.2 Diseño

El diseño de la sincronización temporal está dividido en dos partes. En la primera, estudiamos cómo conseguir la secuencia corta de entrenamiento que el modulador transmitirá antes de cada trama. En la segunda, el diseño se centra en el algoritmo de sincronización.

A) Envío

Para la obtención de la secuencia corta de entrenamiento seguimos el desarrollo explicado en el punto de teoría, 3.2.5. Es decir, introducimos en el bloque IFFT hecho en el apartado 4.2.1.3 la secuencia 3.10. Dicho bloque, calcula la $IFFT_N$, con lo que tendremos que quedarnos con la parte de la salida perteneciente al símbolo periódico, es decir, 16 muestras, y repetirlas para obtener el “short preamble”.

En la figura 4.23 vemos el resultado del cálculo de la $IFFT_{64}$.

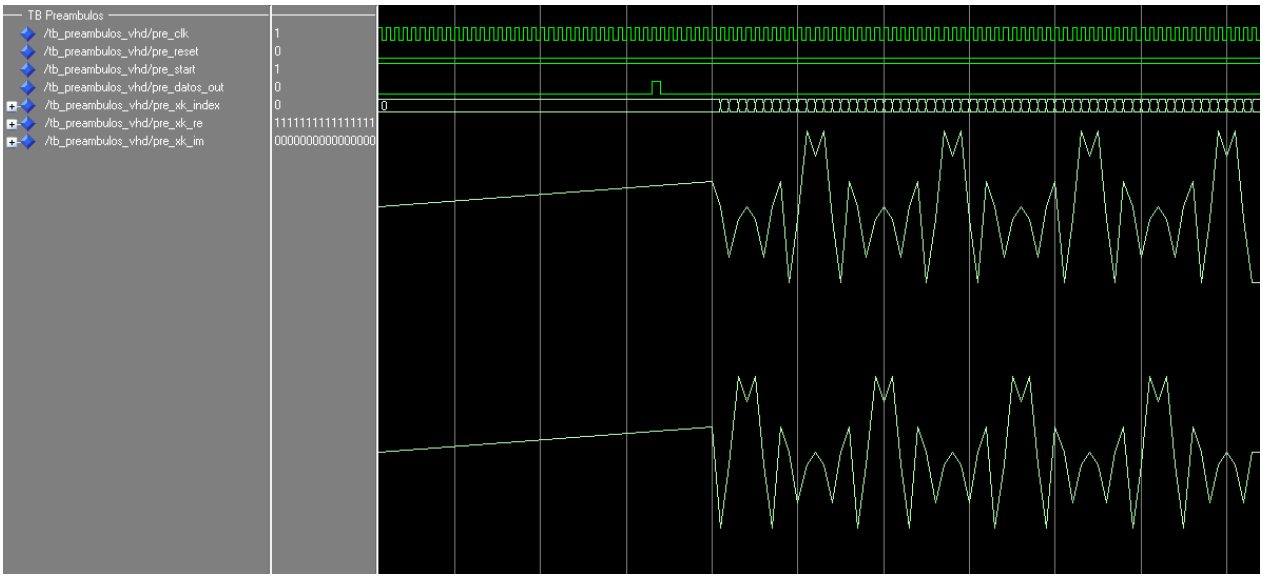


Figura 4.23: *STS calculado por el módulo IFFT*

Si comparamos estas señales con valores dados en el estándar 802.11a representados en la figura 3.15, vemos que como coinciden en forma, salvo que el módulo FFT en VHDL, pone los valores invertidos.

Por tanto, para llevar a cabo la implementación, invertimos los valores a la salida del bloque que calcula la IFFT en la implementación VHDL y, repitiendo las 16 primeras muestras diez veces y añadiendo la muestra 161, obtenemos la secuencia corta de entrenamiento (figura 4.24).

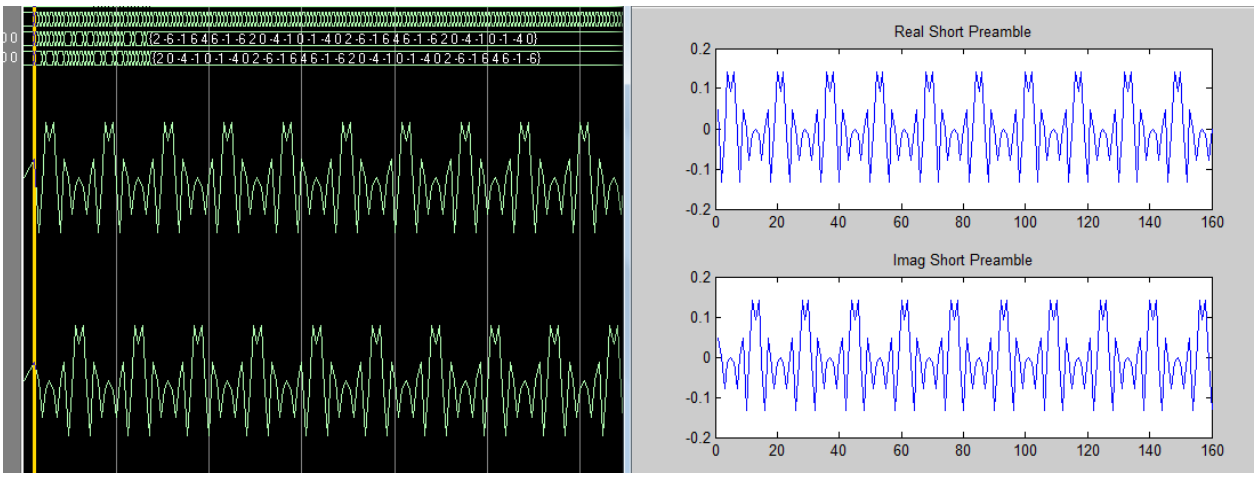


Figura 4.24: *Comparación entre el STS calculado y el STS del estándar 802.11a*

B) Recepción

En la parte del receptor, el diseño se complica un poco más debido a las limitaciones del lenguaje. La idea es que el receptor sólo tenga que estar almacenando 2*L símbolos con los que realizar el cálculo de *Mn* (3.11). De esta manera, obtenemos un nuevo valor de Mn cada vez que recibimos un nuevo valor de la señal, que supondremos del preámbulo de entrada.

Hay que notar, que para simplificar el cálculo de la correlación, almacenaremos los datos en un vector de forma circular, ya que como se trata de una suma, no necesitamos ordenar los vectores de entrada, al ser la suma conmutativa. Además, aunque VHDL permite la utilización de datos en coma flotante, se ha preferido utilizar enteros multiplicados por 1000 (equivalente a tres decimales) por simplificar el cálculo, ya que el objetivo es comparar el valor Mn con un umbral que suele tener un decimal, por lo que no perderemos precisión.

Como en nuestro caso, almacenaremos los valores según llegan al receptor, tendremos que operar el algoritmo con las muestras pasadas, a diferencia de la expresión 3.11 donde se utilizan las muestras futuras para la obtención de la métrica. Esto supondrá que los valores que vayamos obteniendo de Mn tendrán un desplazamiento de L muestras.

Reflejamos estas ideas en un diagrama que representa el algoritmo que queremos llevar a cabo.

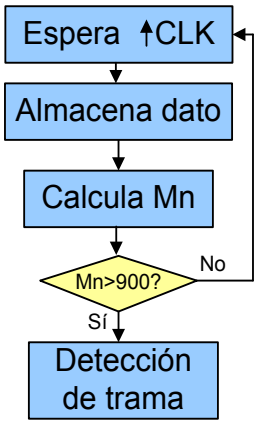


Figura 4.25: *Diagrama del algoritmo de sincronización*

El algoritmo funciona de la siguiente manera. Se encuentra en un bloque secuencial por lo que funciona en los flancos de subida del reloj, donde cada vez que hay uno, realiza las dos operaciones: “Almacena dato” y “Calcula Mn”.

En “Almacena dato” se almacena los datos de la señal de fase y cuadratura en dos vectores circulares (uno para los valores reales y otro para los complejos) de tamaño 32 eliminando el dato almacenado más antiguo en cada uno.

Después opera los vectores, obteniendo un Mn multiplicado por 1000 (para no usar decimales) y compara con el umbral decidido, 900. Si se supera el umbral, implica que el receptor ha detectado el comienzo de una trama OFDM.

Para el cálculo de Mn partimos de la señal ya muestreada y separada en fase y cuadratura, por lo que desarrollamos las ecuaciones 3.11, 3.12 y 3.13 como se indica a continuación.

$$Q(n) = \sum_{m=0}^{L-1}(r_{n+m}^{*} \cdot r_{n+m+L}) = r_n^{*}[0] \cdot r_n[L] + \cdots + r_n^{*}[L-1] \cdot r_n[2L-1] \quad (4.16)$$

$$r[i] = Re\{r[i]\} + j\, Im\{r[i]\} \quad (4.17)$$

$$\begin{aligned} Q(n) = {} & (Re\{r_n[0]\} - j\, Im\{r_n[0]\}) \cdot (Re\{r_n[L]\} - j\, Im\{r_n[L]\}) + \ldots \\ & \ldots + (Re\{r_n[L-1]\} - j\, Im\{r_n[L-1]\}) \cdot (Re\{r_n[2L-1]\} - j\, Im\{r_n[2L-1]\}) \end{aligned} \quad (4.18)$$

Desarrollando y separando la parte real de la imaginaria y, además, particularizando para n=0.

$$\begin{aligned} Re\{Q(0)\} = {} & Re\{r[0]\} \cdot Re\{r[L]\} + Im\{r[0]\} \cdot Im\{r[L]\} + \cdots + \\ & + Re\{r[L-1]\} \cdot Re\{r[2L-1]\} + Im\{r[L-1]\} \cdot Im\{r[2L-1]\} \end{aligned} \quad (4.19)$$

$$\begin{aligned} Im\{Q(0)\} = {} & Re\{r[0]\} \cdot Im\{r[L]\} - Im\{r[0]\} \cdot Re\{r[L]\} + \cdots + \\ & + Re\{r[L-1]\} \cdot Im\{r[2L-1]\} - Im\{r[L-1]\} \cdot Re\{r[2L-1]\} \end{aligned} \quad (4.20)$$

De donde obtener Q(n) es inmediato:

$$Q(n) = (Re\{Q(n)\})^2 + (Im\{Q(n)\})^2 \underset{n=0}{\Longrightarrow} (Re\{Q(0)\})^2 + (Im\{Q(0)\})^2 \quad (4.21)$$

Haciendo el mismo proceso con R(n):

$$R(n) = \sum_{m=0}^{L-1} |r_{n+m+L}|^2 = |r_n[L]|^2 + \cdots + |r_n[2L-1]|^2 \quad (4.22)$$

$$R(n) = Re\{r_n[L]\}^2 + Im\{r_n[L]\}^2 + \cdots + Re\{r_n[2L-1]\}^2 + Im\{r_n[2L-1]\}^2 \quad (4.23)$$

Utilizando las ecuaciones 4.19, 4.20, 4.21 y 4.23 tenemos los valores necesarios para calcular Mn en el instante n.

Si tomamos L=16 el algoritmo de sincronización encontrará su máximo valor en la muestra 32 del “short preamble” de longitud 161 valores. Con lo que tendrá tiempo suficiente para preparar el almacenamiento de los símbolos OFDM.

4.2.5.3 Simulación de la sincronización temporal

Llevamos a cabo la implementación diseñada en el apartado anterior y comprobamos con las simulaciones su funcionamiento, representando en la figura 4.26 los resultados.

Como ya hemos comentado, la parte de sincronización perteneciente al transmisor simplemente enviará la STS al principio de cada trama OFDM, siendo siempre la misma secuencia. Por lo que se gana eficiencia almacenando los 16 valores que se repiten, en una memoria RAM de la FPGA. De modo que a la hora de transmitir, habrá un bloque de control, que lea los estos valores antes del resto de la señal OFDM.

Conectamos a la entrada del bloque que calcula el algoritmo, los valores de la secuencia corta de entrenamiento, y vemos (figura 4.26) como los valores del cálculo de la correlación, comienzan a ser positivos y cada vez más grandes en n=16. Esto es porque en la multiplicación *r(n)*r(n+L)* valdrá distinto de cero cuando obtengamos un valor de *r(n+L)* distinto de 0.

En nuestro caso, y teniendo un L=16, obtendremos el inicio de nuestra rampa de subida en n=16, y el máximo en n=32. Sabiendo en que muestra el receptor detecta idealmente un nuevo paquete OFDM, podremos saber en que momento recibimos la información de los símbolos OFDM.

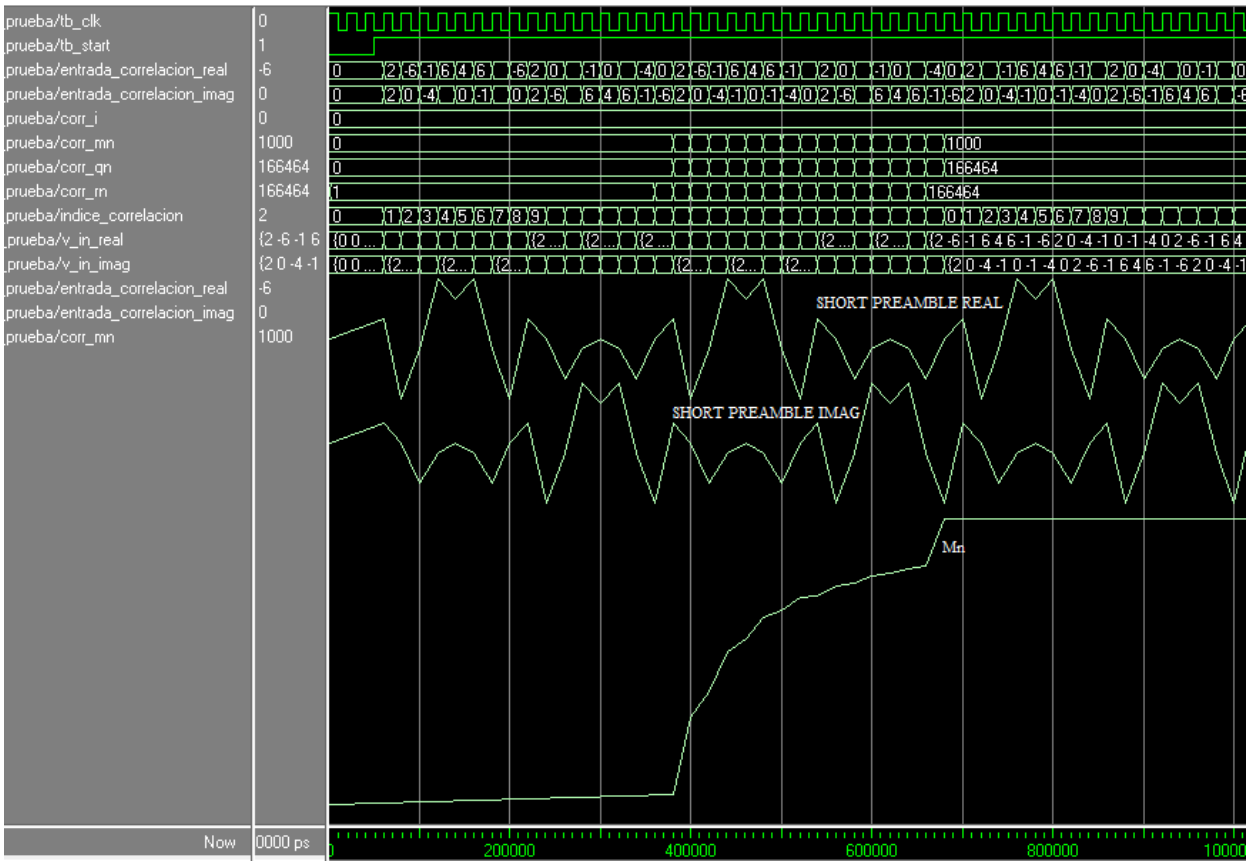


Figura 4.26: *Simulación de la sincronización temporal*

4.3 SISTEMA OFDM COMPLETO

4.3.1 Modulador OFDM

4.3.1.1 Introducción

Llegado a este punto podemos llevar a cabo la implementación de un modulador OFDM, utilizando los diferentes bloques del punto 4.2. De forma que obtengamos un bloque sintetizable que convierta los bits de información en símbolos OFDM, separados en una señal en fase y otra en cuadratura.

Como vimos en el capítulo de teoría, para que el sistema modulador sea completo faltaría llevar a cabo una conversión digital-analógica y una modulación en paso banda para la transmisión. Estas operaciones las llevan a cabo módulos diferentes en un dispositivo SFF SDR, por lo que dejamos para el capítulo 5 el estudio de cómo implementarlo, y en este punto nos centramos en la parte implementada en la FPGA.

En el punto 4.3.1.2 vemos en detalle los problemas de ensamblar los bloques del punto 4.2 y se explican las decisiones tomadas más relevantes a la hora de llevar a cabo la implementación. La explicación del funcionamiento del bloque modulador, se completa con el esquema de conexiones entre bloques (4.3.1.3), con el diagrama de estados (4.3.1.4) y con las simulaciones (4.3.1.5).

4.3.1.2 Diseño

El diseño del modulador OFDM se divide en dos partes. En la primera, se calculan los símbolos OFDM a partir de los bits de información a la entrada del modulador. En el segundo, se diseña la transmisión de los datos, introduciendo además los preámbulos y los prefijos cíclicos.

A) Cálculo de símbolos OFDM

El primer inconveniente que nos encontramos es la descompensación de tiempos entre el bloque de bit2simb y la carga del módulo FFT. Esto se debe a que por cada ciclo de reloj, el módulo bit2simb recibe un bit, con lo que los símbolos se obtienen cada 2 ciclos de reloj en una constelación 4-QAM.

Para evitar este problema, introducimos dos memorias RAM de N índices de tamaño 16 (tamaño de los datos de entrada del modulo FFT). Usaremos una para los símbolos reales y otra para los imaginarios.

Una vez solucionado este inconveniente, vemos el comienzo del comportamiento del sistema. En primer lugar, el modulador debe de permanecer en reposo, hasta que reciba una señal de que están llegando bits. Según vayan llegando los bits, se introducen en el módulo bit2simb que va calculando y poniendo a la salida los símbolos. El modulador, a su vez, controlará que según van saliendo los símbolos, éstos se almacenen en las memorias RAM, en el índice oportuno.

Cuando hayamos calculado N símbolos, el modulador configurará la FFT de forma que calcule la IFFT de los datos de la RAM, para ello seguirá los pasos descritos en el diseño del módulo FFT (4.2.1.3).

En este punto nos encontramos con el mismo inconveniente que en la transformación de bits a símbolos, ya que queremos hacer un modulador que calcule hasta 100 símbolos OFDM para que sean transmitidos a la vez. Por lo que cada vez que el modulador calcule un símbolo OFDM, lo almacenará para posteriormente transmitirlos todos seguidos en una trama.

Necesitamos por tanto una memoria RAM que almacene hasta 6400 valores, para lo que necesitaremos un bus de dirección de memoria de 13 bits (2^{13}=8192>6400).

En cuanto al tamaño, los datos de salida del bloque IFFT son de 27 bits, sin embargo en este punto se ha utilizado un tamaño de palabra igual al de entrada (16 bits) tanto para probar el modulador con el demodulador. Ya que además, se ha podido comprobar que en la conversión no se perdía información.

B) Transmisión de los símbolos OFDM

Una vez tenga almacenados los 100 símbolos OFDM calculados, el sistema ya puede transmitirlos todos. Hay que notar, que los datos de salida de la IFFT están en el orden inverso, por lo que aprovechamos a su vez, el uso de las memorias para transmitirlos en orden. El transmisor pasa por tres fases.

En la primera, transmite los preámbulos calculados en el punto 3.2.5 y que servirán para la sincronización en el receptor. Los preámbulos cortos no se calculan cada vez, ya que como siempre con los mismos se almacena como una constante. Como es periódico con almacenar 16 muestras es suficiente.

A continuación, el sistema de control del modulador, va leyendo los datos almacenados en las memorias RAM, de forma que, en cada nuevo símbolo a transmitir, primero lee los últimos 8 datos (N/64) para transmitir el prefijo cíclico, y a continuación, el símbolo OFDM.

Cuando termina de enviar todos, reinicia variables y vuelve al estado de *Esperando_bits* para comenzar con otra trama. El funcionamiento paso a paso del modulador lo vemos en el punto 4.3.1.4.

4.3.1.3 Conexionado de bloques

Siguiendo estas pautas, llevamos a cabo el diseño del bloque modulador. Este bloque visto desde fuera, sólo necesita, además de las entradas de reloj (*clk*) y *reset*, la entrada para los bits de información; y como salida, las señales en fase y cuadratura.

Pero además añadimos una entrada y una salida para llevar a cabo la sincronización del envío de bits. La salida *ready* indica que el bloque está listo para recibir bits y la entrada *Rx_bits* es una entrada lógica que se activa a la vez que se introducen bits en el modulador.

Vemos en la figura 4.27 una representación del modulador, en la que añadimos también la señal *Tipo_QAM* para una futura ampliación.

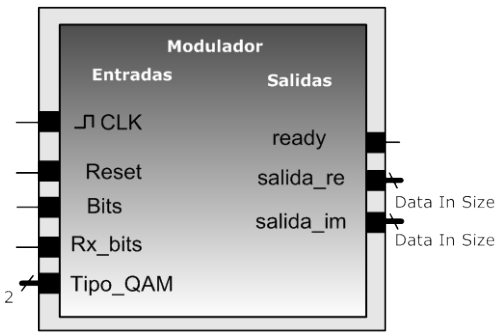


Figura 4.27: *Bloque modulador.*

- **Señales de entrada:**

  - *CLK*: reloj del módulo
  - *Reset*: señal de reset activa a nivel alto
  - *Bits*: bits de entrada
  - *Rx_bits*: señal de control
  - *Tipo_QAM*: 4, 8, 16 ó 64 QAM

- **Señales de salida:**

  - *ready*: indica que se están poniendo datos a la salida
  - *salida_re*: símbolo real
  - salida_im: símbolo imaginario

A partir de la "caja negra" del modulador, podemos llevar a cabo la interconexión entre los bloques del punto 4.2, ya que la salida de algunos de estos bloques serán la entrada de los otros. En primer lugar, la entrada bits del modulador, irá conectada directamente a la entrada del bloque *bits2simb* que cada dos periodos de reloj que está recibiendo bits, pone a la salida el símbolo real e imaginario asociado a la constelación.

Estos símbolos se almacenan directamente en las memorias RAM debido a la descompensación temporal entre la salida de *bits2simb* y la entrada de *modulo_ifft*. Tanto en esta memoria como en la de los símbolos OFDM utilizamos una memoria para los símbolos reales y otra para los imaginarios.

La salida de esta primera memoria, se conecta directamente a la entrada de *modulo_ifft*, y la salida de este modulo se conecta directamente a la entrada de las memorias RAM de los símbolos OFDM donde se almacenan tantos símbolos OFDM como se quieran transmitir en una trama.

En la figura 4.28 vemos el esquema del conexionado interno del bloque *modulador*. Para simplificar el esquema sólo se representan las conexiones entre bloques y, las entradas y salidas del *modulador*. El resto de señales están controladas por la lógica de control del bloque *modulador* (señales en azul en el esquema). Todos los bloques comparten el mismo reloj.

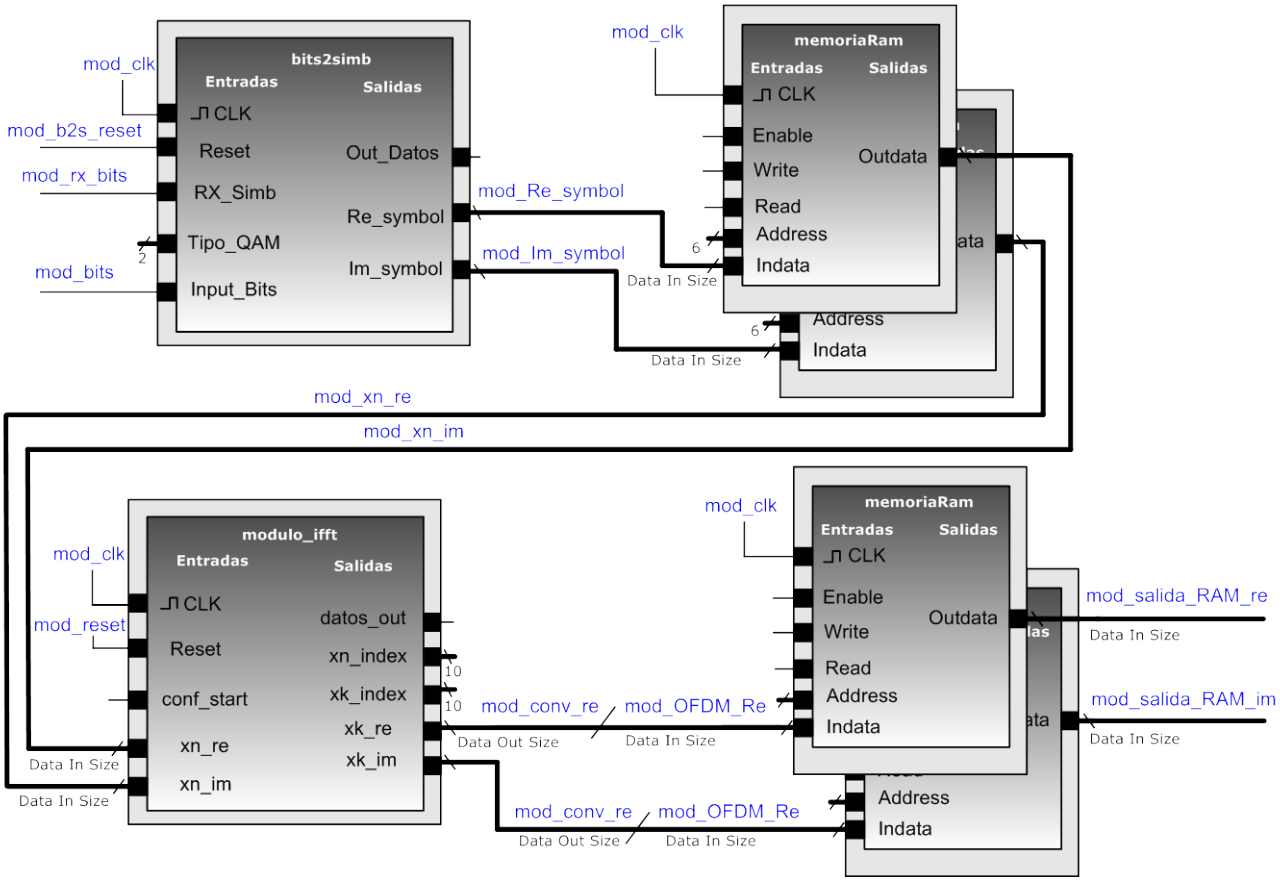


Figura 4.28: *Conexionado del modulador*

Cabe destacar que la señal de *reset* de *bits2simb* también está controlada por la lógica de control ya que entre el cálculo de un símbolo OFDM y otro se hace un reset para asegurarnos que no se ha quedado con información del símbolo anterior.

Por último, representamos en la figura 4.29 la elección de la salida del modulador mediante un multiplexor en función del estado indicado en la señal *mod_estado*. Esto se debe a que sólo pondremos a la salida del bloque los símbolos OFDM calculados cuando corresponda, poniendo las salidas a cero mientras se estén calculando símbolos y utilizando el valor almacenado de los preámbulos en una constante para el envío de los mismos. Esta síntesis está llevada a cabo indirectamente, especificando la salida utilizada en cada estado.

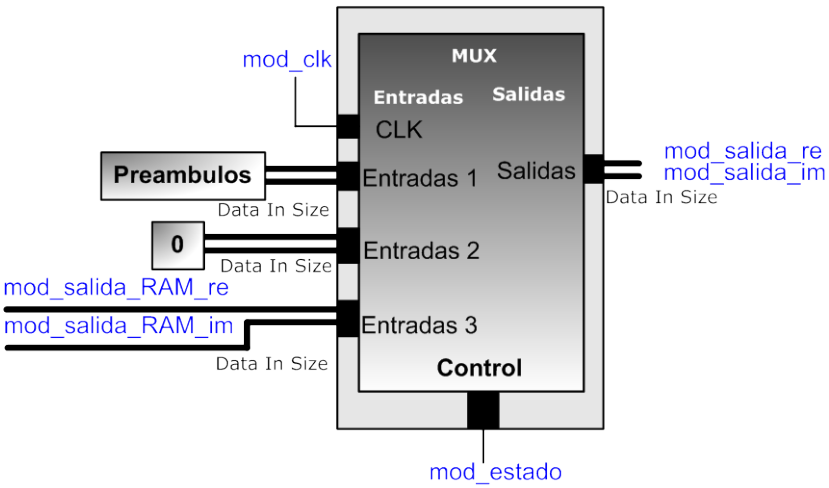


Figura 4.29: *Elección de salidas en el modulador*

4.3.1.4 Máquina de Estados

Tras haber analizado el diseño del modulador y haber hecho un esquema con las señales necesarias, representamos el diagrama de estados del modulador. Para simplificar su comprensión, en el esquema de la figura 4.30 se representan sólo las transiciones entre los estados y los contadores que producen un cambio de estado. Estos contadores se modifican dentro de cada estado según las operaciones que realiza cada uno. El contador 'i' se utiliza para los N datos de un símbolo OFDM, usándose de forma creciente o decreciente según convenga, y el contador 'Nofdm' para el número de símbolo en la trama. El resto de operaciones que se llevan a cabo dentro de cada estado se explica de manera general a continuación.

En primer lugar, el modulador entra en el estado de Inicio, donde inicializa todas las señales internas y salidas. Cuando termina, pasa a esperar la recepción de los bits a transmitir, en el estado *Rx_bits*. Para indicar que está listo a la fuente exterior que envía la información, activa la señal *mod_ready*.

En este estado, según recibe los bits, los asocia a la constelación 4-QAM (figura 4.14) y almacena los símbolos de la constelación en la memoria RAM *memoriaRAMsimb*. Para cada símbolo almacenado, entra dos veces en estado, ya que necesitamos esperar a que lleguen dos bits al bloque bits2simb para obtener un símbolo.

Utilizando un contador, que se usa además para indicar la posición de memoria del símbolo de la constelación, se detecta cuando hemos almacenado N símbolos. En ese momento se pasa al estado de *Carga_IFFT*.

En el estado de *Carga_IFFT* se configura el *modulo_ifft* como se explicó en el punto 4.2.1.3, y se cargan los datos en dicho módulo leyéndolos de la memoria RAM. Cuando se terminan de cargar los datos se cambia al estado *Calculando_IFFT* que espera la activación de la señal *ifft_datos_out*, como indicador de que se ha terminado el cálculo de la IFFT y se va a comenzar a poner los datos a la salida, por lo que pasa al estado *Almacena_Simb_OFDM* que se encargará de almacenarlos.

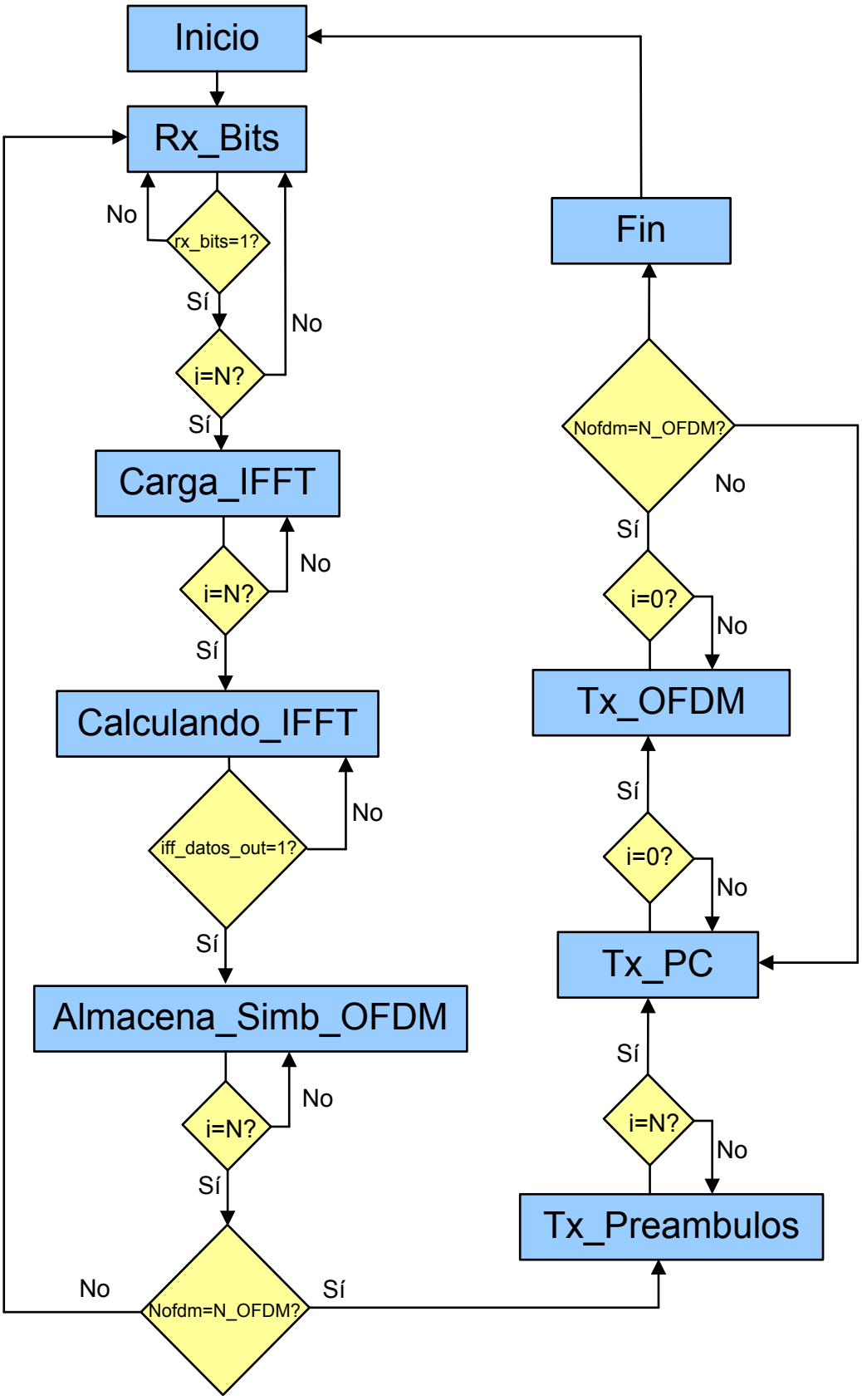


Figura 4.30: *Diagrama de estados del modulador*

En *Almacena_Simb_OFDM* se escribe el símbolo OFDM calculado en la memoria RAM de símbolos OFDM (de tamaño N * N_OFDM). Este estado lleva dos contadores, uno para saber la posición en la trama del símbolo calculado (Nofdm) y otro para indicar la dirección de memoria de los N datos de un símbolo OFDM. Cada vez que almacena un símbolo OFDM, comprueba si es el último y en tal caso pasa a transmitir los datos. En caso contrario, vuelve al estado de *RX_Bits*, activando la señal de *mod_ready* para recibir más bits que codificar.

Cuando se han calculado los N_OFDM símbolos que se transmiten en una trama, el modulador comienza a transmitir los datos. El primer estado por el que pasa es el de *TX_Preambulos* donde se transmiten los preámbulos cortos a modo de cabecera de sincronización de la trama.

Al terminar de transmitir los preámbulos, comienza a transmitir los símbolos OFDM leyéndolos de la memoria RAM OFDM. Para hacerlo, primero transmite el prefijo cíclico en el estado TX_PC transmitiendo las últimas 8 muestras del símbolo.

A continuación del prefijo cíclico, transmite el símbolo, y así sucesivamente con los N_OFDM símbolos almacenados. Cuando envía el último, pasa al estado *Fin* que reinicia todos los contadores. De *Fin* pasa directamente a Inicio, y de Inicio a *Rx_bits*, donde se queda a la espera de recibir nueva información que enviar y así reanudar de nuevo todo el proceso.

4.3.1.5 Simulaciones

Llevando a cabo las especificaciones explicadas en los puntos anteriores, implementamos en código VHDL el modulador y simulamos en Modelsim su comportamiento, que vemos paso a paso.

Con el diseño llevado a cabo pasamos a probar su funcionamiento, creando un banco de pruebas que posteriormente utilizaremos también para el demodulador en el punto 4.3.2.

- **Datos de las simulaciones:**

  - Tamaño de trama de 100 símbolos OFDM
  - N=64
  - PC: N/8
  - Sincronización mediante “short preamble”
  - Frecuencia de trabajo 50 MHz
  - Una secuencia de 128 bits

En primer lugar el bloque de pruebas hace un reset para asegurarse de que se inicia correctamente el modulador. Después, espera a que la señal de que el modulador está listo para recibir bits (mod_ready) se active, y en ese momento, comienza a enviarle los bits de información activando a la vez la señal *rx_bits*.

Vemos en el primer cronograma (figura 4.31) como se inicia el modulador y como asocia los bits recibidos en mod_bits a los símbolos de la constelación 4-QAM en mod_re_symbol y mod_im_symbol y almacena los símbolos en la memoria RAM mediante la activación de la señal *mod_ram_write*.

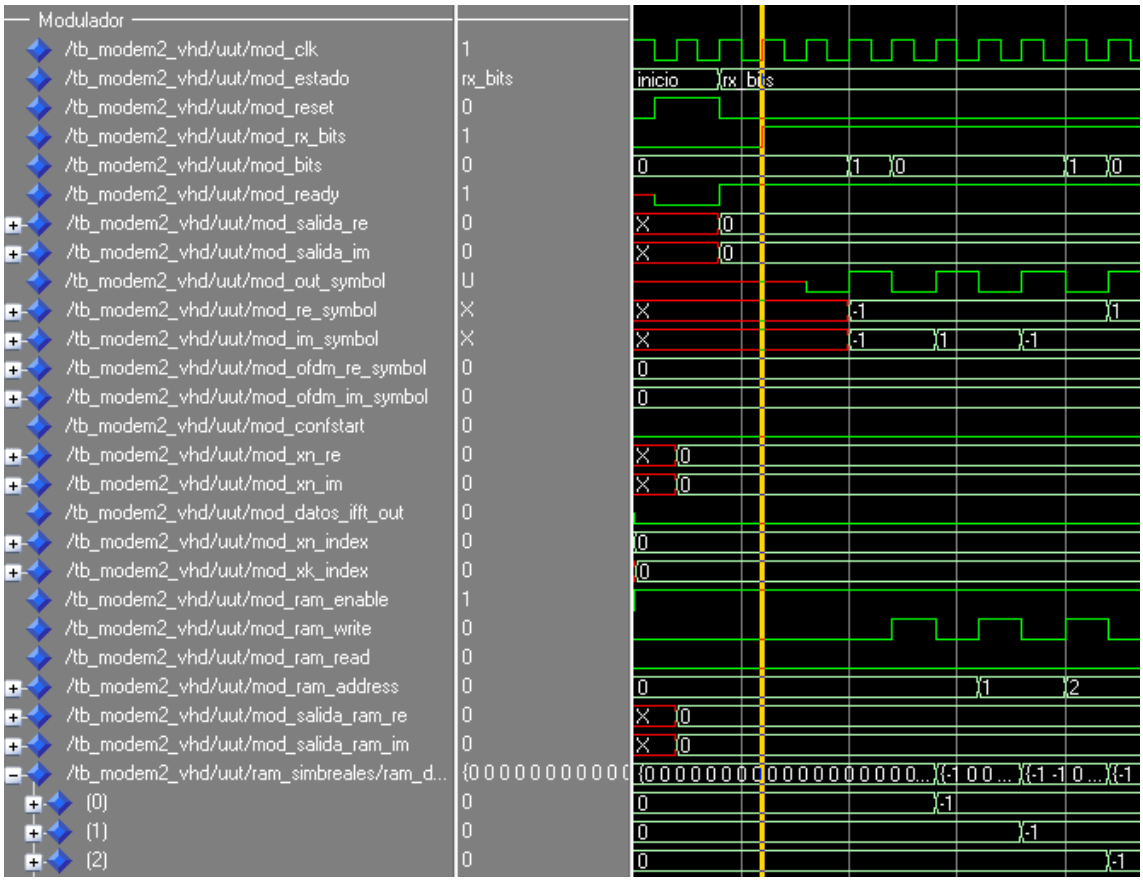


Figura 4.31: *Inicio del modulador y almacenamiento de los símbolos de la constelación*

En la figura 4.32 vemos como termina de almacenar los 64 símbolos de la constelación y pasa a cargarlos en el *modulo_ifft* activando la señal *ram_read*. Esta señal permanece activa toda la carga de datos, ya que la temporización de la lectura de la RAM y de la carga en *modulo_ifft,* coinciden.

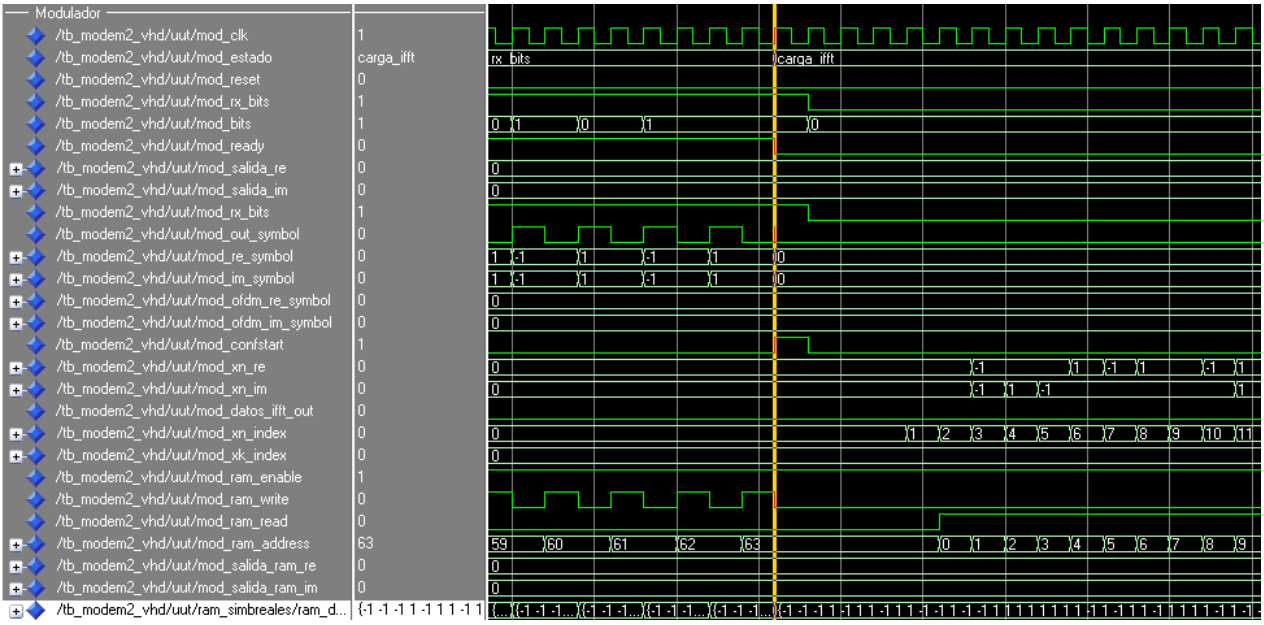


Figura 4.32: *Fin de almacenamiento y carga de los símbolos de la constelación*

Cuando termina de cargar los datos, el bloque *modulo_ifft* comienza a calcular la IFFT de los símbolos de la constelación.

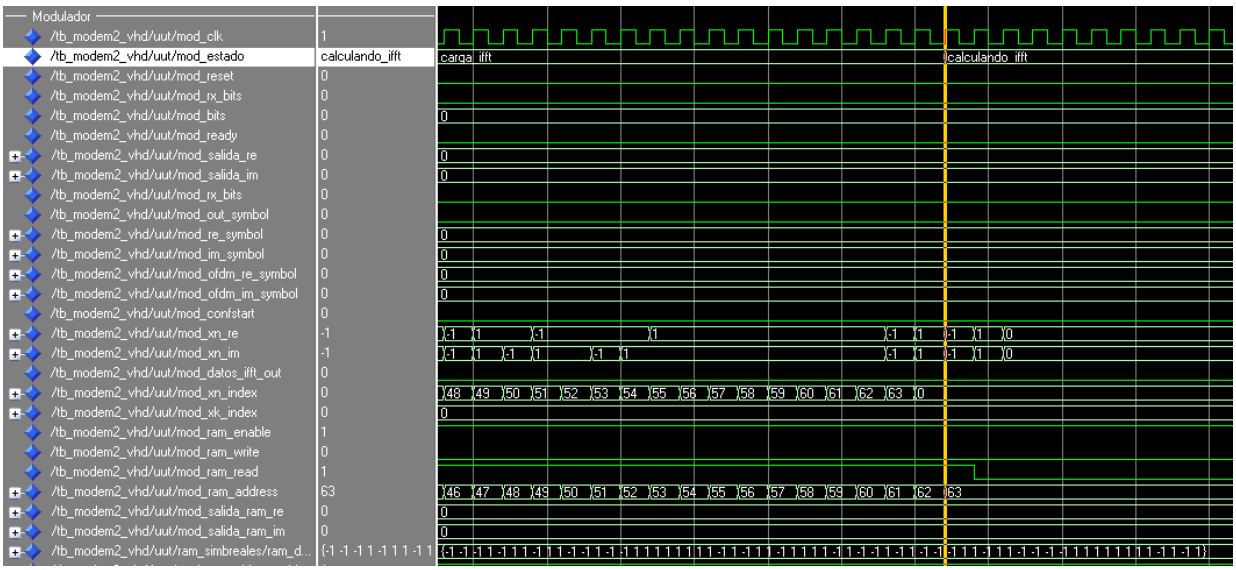


Figura 4.33: *Fin de carga y comienzo del cálculo de IFFT.*

Cuando acaba se activa la señal *ifft_datos_out* y pasa al estado *Almacena_Simb_OFDM* donde espera a los datos calculados. Según salen se van almacenando en la otra memoria RAM usada para los símbolos OFDM, como vemos en la figura 4.34.

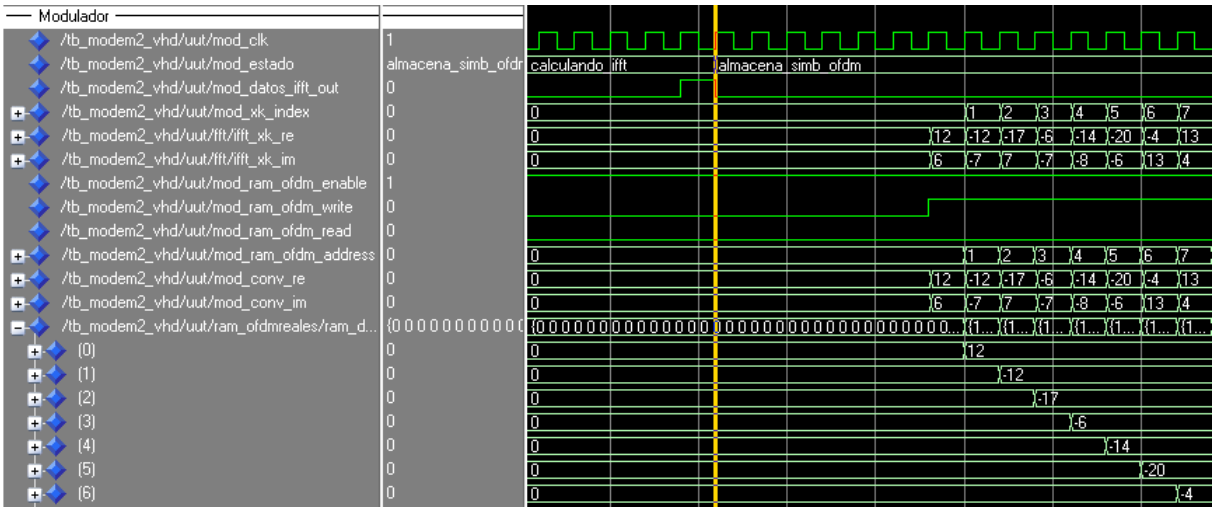


Figura 4.34: *Almacenamiento del símbolo OFDM*

Este proceso se repite para los 100 símbolos contenidos en una trama. En el cronograma de la figura 4.35 vemos el cálculo de dos símbolos y el comienzo de un tercero.

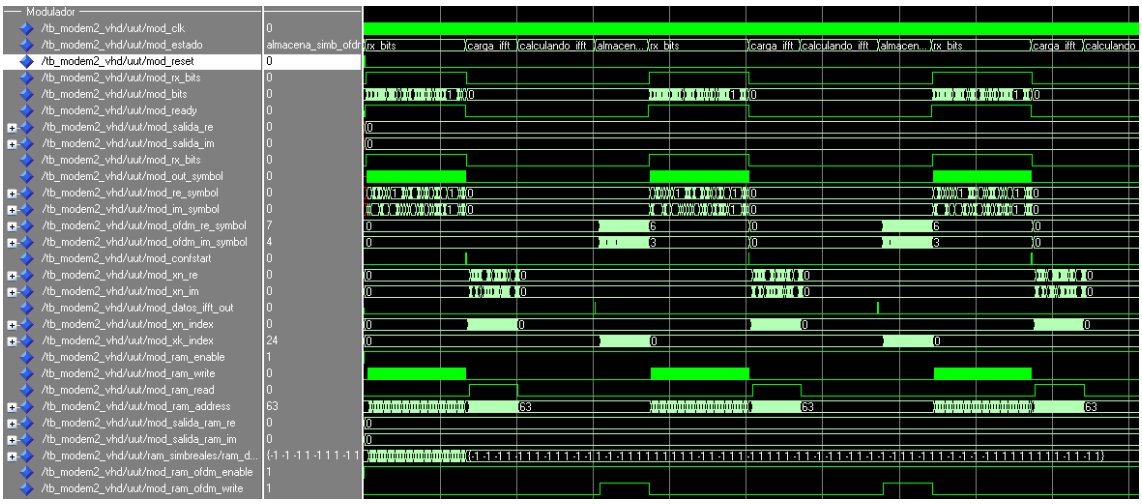


Figura 4.35: *Cálculo de varios símbolos OFDM*

Este proceso se repite hasta que alcanzamos los 100 símbolos OFDM, con lo que el último dato OFDM se almacena en la posición 6399.

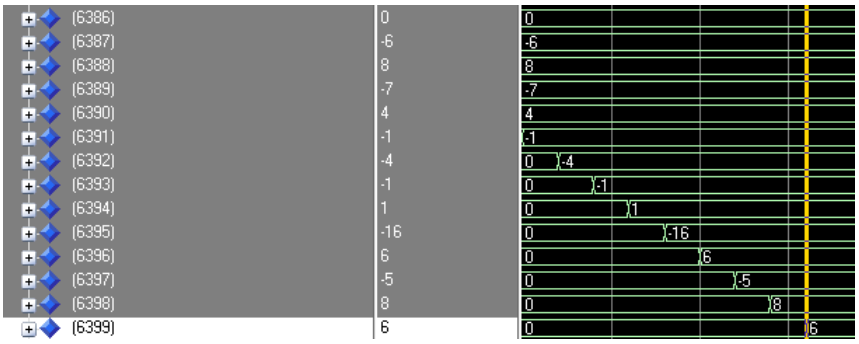


Figura 4.36: *-Fin de almacenamiento de los símbolos OFDM*

Con el almacenamiento del símbolo 100, se termina el cálculo de los símbolos OFDM y comienza la transmisión de datos.

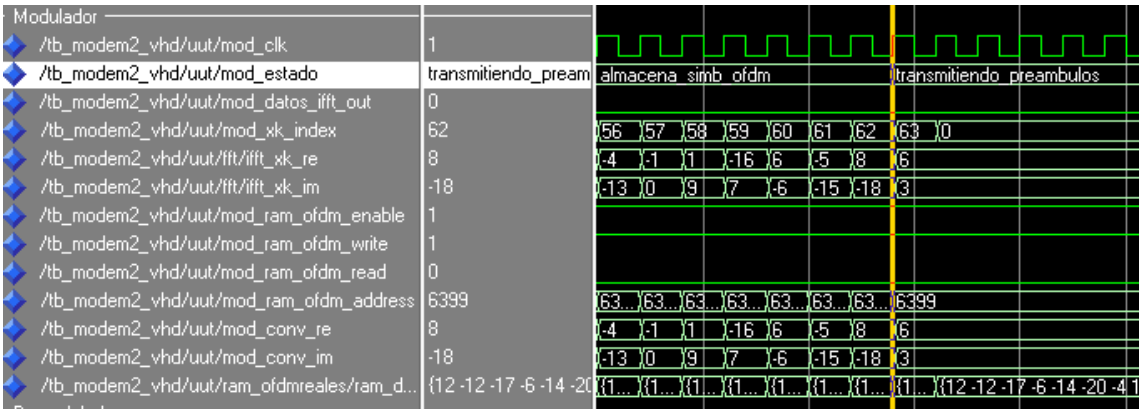


Figura 4.37: *Inicio de almacenamiento de los símbolos de la constelación*

Lo primero que se transmite es la secuencia calculada en el punto 4.2.5.2, conocida como preámbulo corto. Para representarlas de forma que se parezcan a las señales de transmisión, mediante Modelsim, interpolamos la señal de forma lineal y conseguimos el efecto de una señal analógica.

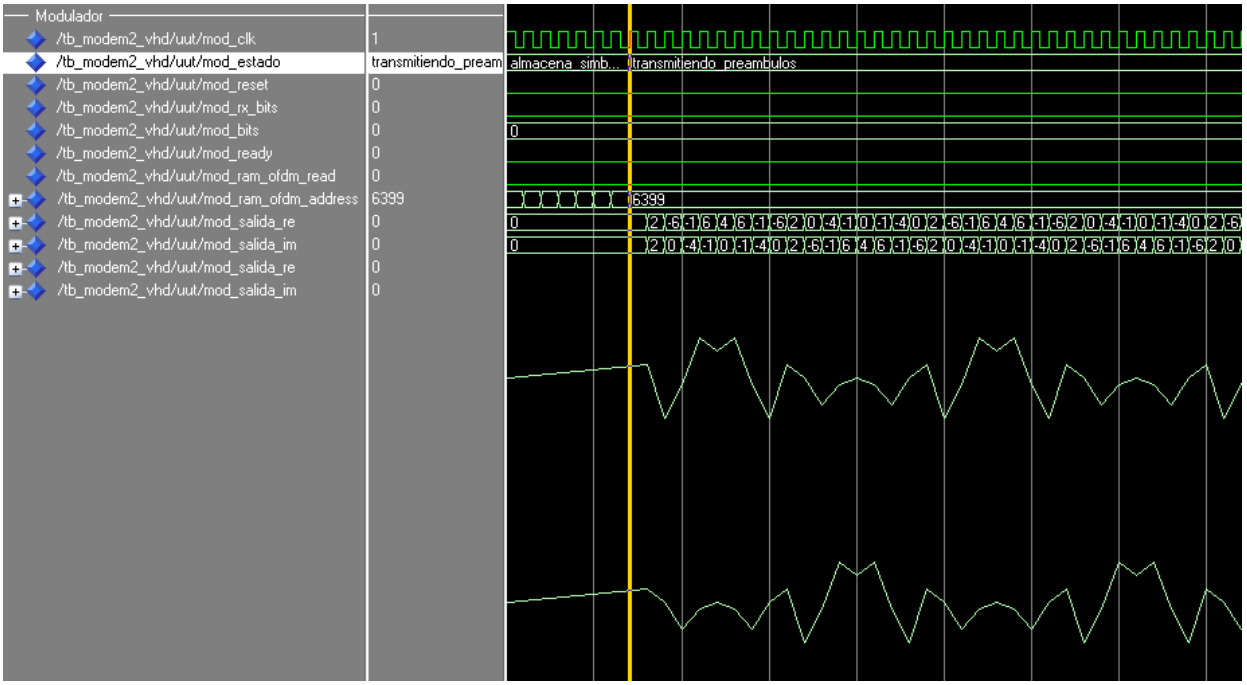


Figura 4.38: *Transmisión del preámbulo.*

Tras los preámbulos se comienza a enviar los símbolos OFDM.

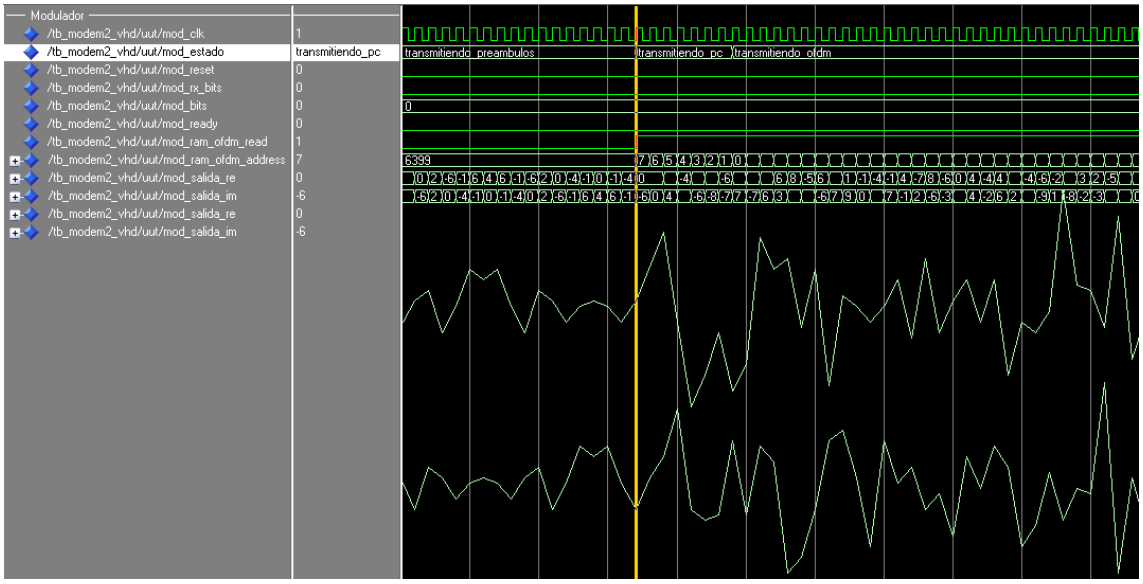


Figura 4.39: *Transmisión. Fin de preámbulos e inicio de símbolos OFDM.*

Cada símbolo OFDM se transmite con su prefijo cíclico, como vemos marcado en la figura 4.40. Este proceso se repite para los 100 símbolos OFDM.

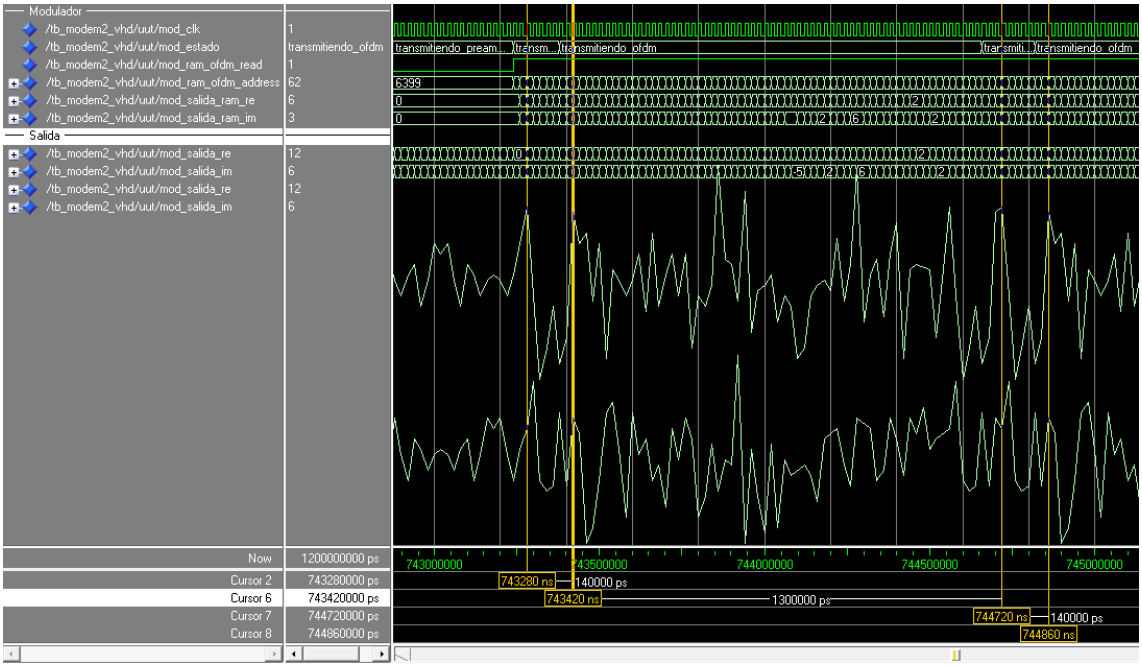


Figura 4.40: *Símbolos OFDM con PC.*

En la figura 4.41 vemos una visión completa de la trama OFDM enviada.

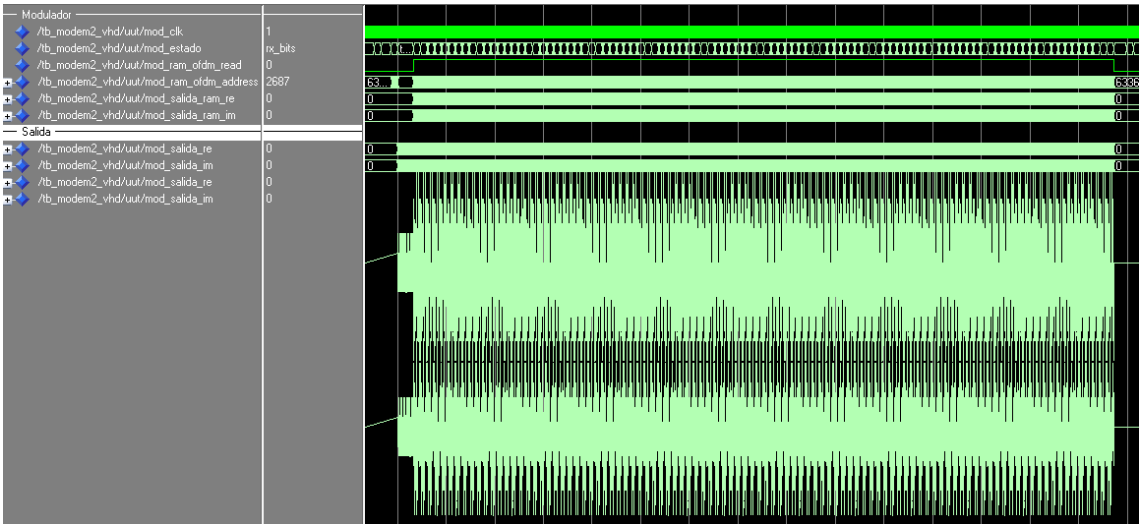


Figura 4.41: *Trama OFDM completa.*

4.3.2 Demodulador OFDM

4.3.2.1 Introducción

Una vez montado y probado el bloque modulador, pasamos a utilizar los bloques necesarios del capítulo 4.2 para llevar a cabo la implementación de un bloque demodulador OFDM. Con el demodulador hecho, lo probaremos conectándolo a la salida del bloque modulador, de forma que podamos comprobar la información recibida con la que se envió.

Al igual que nos ha ocurrido con el modulador, el sistema receptor OFDM se completaría con un bloque que antes de la entrada a nuestro demodulador, separe la señal analógica modulada en una señal en fase y otra en cuadratura muestreadas. Por tanto, en este capítulo, nos centraremos en el diseño a partir de estas dos señales para recuperar los bits de información.

Seguiremos la misma estructura que en el punto 4.3.1 (Modulador). Así, en el punto 4.3.2.2 veremos la explicación de las decisiones tomadas en el diseño del demodulador y en los puntos siguientes, profundizaremos en su funcionamiento con la explicación del esquema del conexionado de bloques (4.3.2.3), con el diagrama de estados (4.3.2.4) y con las simulaciones (4.3.2.5).

4.3.2.2 Diseño

El objetivo del bloque, es obtener de la manera más eficiente posible, los bits de información que fueron enviados. El punto de partida que tomamos en este apartado es la recepción de dos señales (fase y cuadratura) y muestreadas a la frecuencia de trabajo de la FPGA.

Estas operaciones las realizan los otros módulos del dispositivo SFF SDR que no son la FPGA, el módulo de conversión de datos y el módulo de RF, como veremos en el capítulo 5.

A) Sincronización y almacenamiento

En primer lugar, tenemos que tener en cuenta a la hora de escoger el tamaño de palabra de las señales de entrada del demodulador, que estará limitada por el bloque *modulo_fft* a 16 bits.

Para agilizar la sincronización, ésta se lleva a cabo mediante una función VHDL que lleve a cabo el algoritmo de sincronización (3.11) desarrollado en el punto 4.2.5, evitando así crear un que conectar a la recepción. Por tanto, el primer bloque del demodulador será la memoria RAM que estará controlado por el control del demodulador, y que almacenará los símbolos OFDM en el momento indicado por la sincronización.

Como recibiremos una trama de hasta 100 símbolos OFDM con sus prefijos cíclicos, tendremos que almacenar sólo la información de los símbolos OFDM, desechando los prefijos cíclicos. Por tanto, antes de pasar por el bloque FFT, el demodulador irá almacenando todos los símbolos de la trama y para ello, utiliza dos bloques de memoria RAM, una para los símbolos reales y otra para los imaginarios, donde almacenaremos hasta N*100, es decir, hasta 6400 datos.

B) Cálculo de FFT y obtención de bits

Una vez almacenados los símbolos OFDM reales e imaginarios, la demodulación llevada a cabo por la FFT es inmediata, y simplemente bastará con cargar los datos en el módulo FFT. Como ya ocurrió en el modulador, el bloque que calcula la FFT, saca el resultado de invertido, por lo que aprovecharemos que tenemos la información almacenada, para cargarla invertida, y obtener la secuencia de bits en el orden en que se enviaron al modulador.

La salida del bloque que calcula la FFT se conectará al bloque simb2bits, obteniendo de manera directa los bits decodificados. Como ya comentamos en la introducción, utilizaremos el bloque modulador para generar las señales en fase y cuadratura y así poder comprobar directamente que los bits recibidos coinciden con los enviados.

4.3.2.3 Conexionado de bloques

En este punto pasamos a representar el conexionado interno entre bloques del demodulador. Para ello, primero vemos la "caja negra" que queremos del demodulador.

La idea es tener un módulo que reciba las señales en fase y cuadratura y ponga a la salida los bits de información contenidos en los símbolos OFDM recibidos. En este caso no debe de haber sincronización externa para recibir la información como ocurría en el modulador, ya que el demodulador no puede saber en qué momento le llega la información, para lo que existe el algoritmo de sincronización interno. Sin embargo, si que interesa tener sincronización externa a la hora de sacar los bits de información de salida, ya que puede ser muy útil para que el sistema receptor gestione la información a tiempo real a nivel de aplicación.

Por tanto, diseñamos el bloque demodulador, con las entradas típicas de reloj (*clk*) y *reset*, *Tipo_QAM* para una futura ampliación donde elegir la constelación y las señales de fase y cuadratura. Y a la salida con los bits de información le añadimos otra lógica, para indicar que se están poniendo bits a la salida.

Vemos en la figura 4.42 la representación de bloque *demodulador*.

- **Señales de entrada:**

  - *CLK*: reloj del módulo
  - *Reset*: señal de reset activa a nivel alto
  - *entrada_re*: parte real de la señal de entrada
  - *entrada_im*: parte compleja de la señal de entrada
  - *Tipo_QAM*: 4, 8, 16 ó 64 QAM

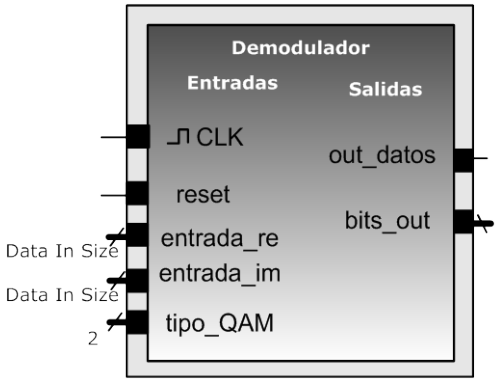


Figura 4.42: *Bloque demodulador.*

- **Señales de salida:**

  - *out_datos*: indica que se están poniendo datos a la salida
  - *bits_out*: bits

A partir de las entradas y salidas, llevamos la interconexión de los bloques que compondrá el demodulador. En primer lugar, necesitamos almacenar los símbolos OFDM que se reciben a través del receptor, por lo que la entrada estará directamente conectada a una memoria RAM para los símbolos OFDM, y la lógica de control se encargará de no almacenar el prefijo cíclico y de sincronizar la recepción.

Cuando se hayan almacenado los símbolos de una trama completa, el sistema tiene que empezar a hacer la FFT cada N datos, por lo que la salida de la memoria RAM la conectamos al bloque *modulo_fft*, y el control se encargará de sincronizar la carga de datos.

Cuando termina de hacer la FFT, el bloque *modulo_fft* pone en la salida los datos calculados. Estos datos se conectan directamente al bloque *simbolos2bits*, que de forma combinacional pone a la salida del demodulador la decisión llevada a cabo sobre los símbolos.

Representamos en la figura 4.43 las conexiones entre los bloques internos del demodulador (en azul las señales del demodulador). Las entradas y salidas de los bloques sin conectar, están conectadas a la lógica de control del demodulador.

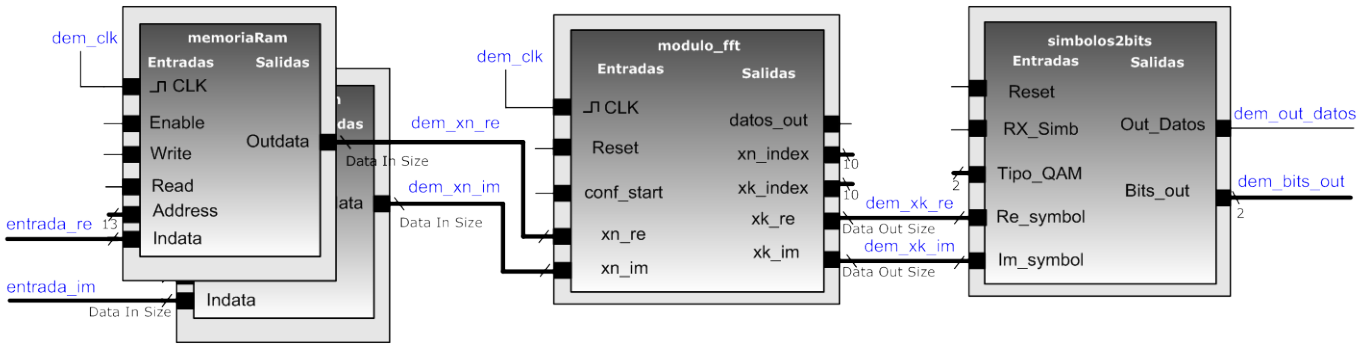


Figura 4.43: *Conexionado del demodulador*

4.2.3.4 Máquina de estados

Partiendo de las ideas explicadas en el diseño, y junto con el conexionado de bloques del punto anterior realizamos el diagrama de estados de la lógica de control del demodulador.

En el esquema de la figura 4.44 representamos las transiciones entre estados, explicando en este punto, las acciones llevadas a cabo en cada estado. Utilizamos el contador 'i' de manera similar a como se usó en el modulador, pero en este caso, usamos el contador *rx_cnt* para la cuenta de los N*N_OFDM datos de los símbolos OFDM en lugar de utilizar dos contadores, ya que el valor de este contador, nos es muy útil para la utilizar la dirección de acceso a memoria.

Al arrancar, el demodulador comienza en el estado *Inicio*, en el que se inicializan las variables y salidas. Y pasa directamente a Sincronización, que se encarga de operar los valores de las señales de entrada para detectar la secuencia corta de entrenamiento que encabeza la trama.

Cuando el resultado de este algoritmo (M_n) es mayor de 0.9 (900 en la implementación ya que está multiplicado por 1000), detecta la secuencia y pasa a *Espera_simbolos*, que espera a que comiencen a llegar los símbolos OFDM. El valor 129 viene de que el algoritmo de sincronización detecta la muestra 32, y los preámbulos tienen una longitud de 161 muestras (161-32=129).

En este punto entra en un bucle que dura el número de símbolos de la trama OFDM y en el que pasa continuamente por los estados *Quita_PC* y *Almacena*. En este primer estado se desechan los datos, ya que pertenecen al prefijo cíclico, y en el segundo se almacenan de forma continua en la memoria RAM del demodulador, de tamaño N*N_OFDM.

Cuando se terminan de recibir los datos, se arranca en el estado *Inicia_FFT* el bloque *modulo_fft* y se espera a que esté listo para cargar los datos de N en N, en *Carga_FFT*. Cuando se terminan de cargar los datos, se espera la salida de la FFT en Calculando_FFT.

Al terminar, el bloque *modulo_fft* saca los resultados que, según van saliendo, pasan por el bloque *simbolos2bits* que lleva a cabo la decisión sobre la constelación 4-QAM, y pone los bits a la salida en grupos de dos indicándolo mediante la señal *datos_out*.

Una vez que se acaban de salir los datos obtenidos de la FFT, lo que se detecta con la señal *xk_index*, se comprueba si se ha acabado de decidir sobre todos los datos recibidos. Si es así, se pasa al estado *Fin* que reinicia el demodulador. En caso contrario, se vuelve a *Inicia_FFT* para volver a hacer la FFT y la decisión de otros N datos del símbolo OFDM.

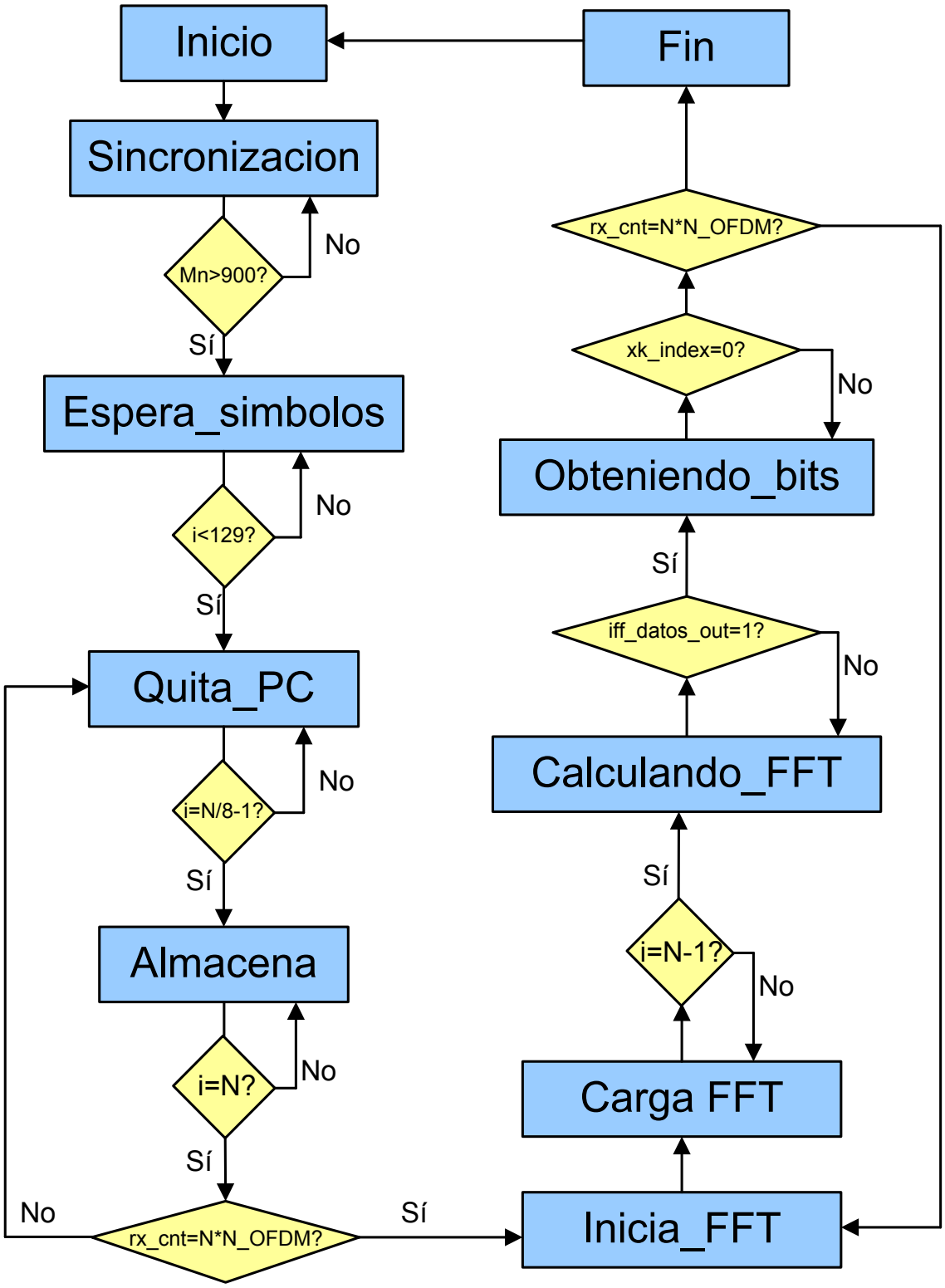


Figura 4.44: *Diagrama de estados del demodulador*

4.2.3.5 Simulaciones

Para llevar a cabo la simulación, utilizamos un bloque de pruebas que conecta la salida del modulador a la entrada del demodulador, quedando de la siguiente forma:

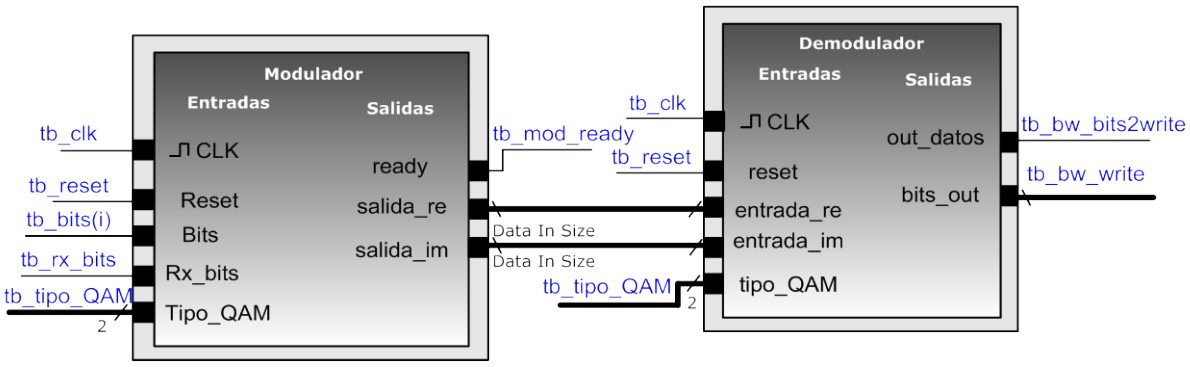


Figura 4.45: *Diagrama del programa de pruebas modem*

En el código de pruebas de la simulación, las salidas del demodulador se conectan a un módulo VHDL que simplemente escribe los datos que salen en un fichero, para poder comprobar que coinciden con los introducidos por *tb_bits(i)*.

Los datos de las simulaciones que usa el modulador (N=64, N_OFDM=100, etc) también los conoce el demodulador.

En primer lugar, tras iniciarse el demodulador pasa al estado de sincronización donde ejecuta el algoritmo de sincronización con las señales de entrada. En el cronograma de la figura 4.46 vemos como el valor Mn va aumentando, hasta la muestra 32, en el que tiene un valor superior a 900, por lo que pasa al estado de *Espera_simbolos*.

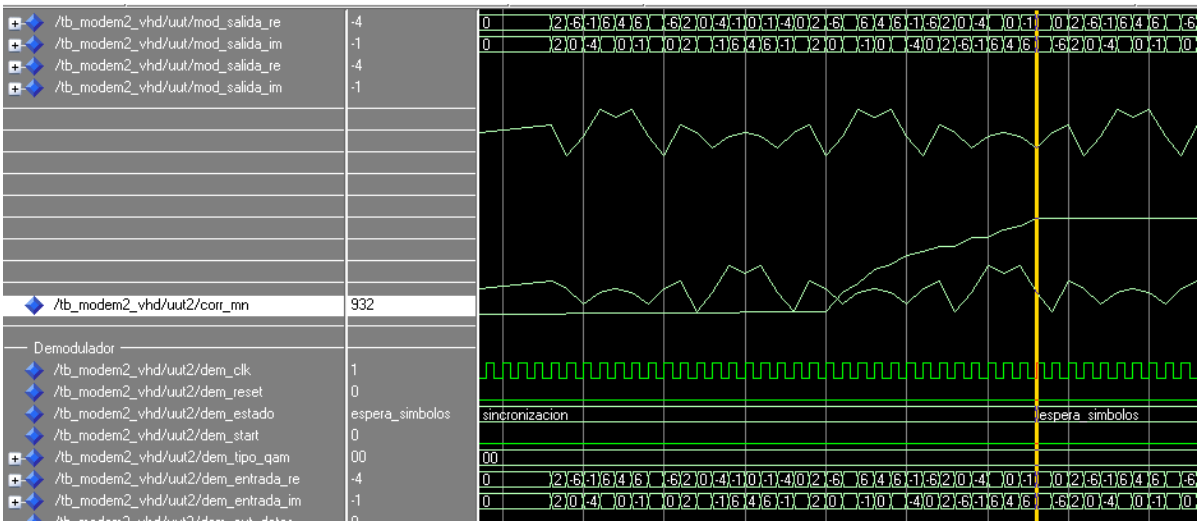


Figura 4.46: *Sincronización del demodulador.*

En el siguiente cronograma vemos como la recibir la señal y detectar el comienzo de los símbolos OFDM, el demodulador comienza a almacenar la información recibida quitando el PC.

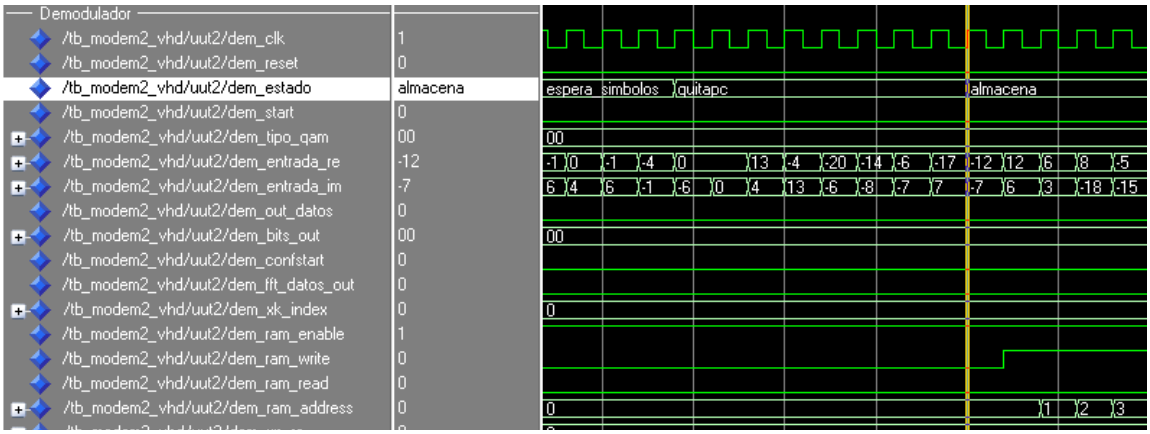


Figura 4.47: *Comienzo del almacenamiento de símbolos OFDM.*

Esta operación se repite tantas veces como símbolos OFDM hay en la trama. En la figura 4.48 podemos ver como se repite el proceso, activándose la señal *dem_ram_write* sólo en los periodos de *Almacena*.

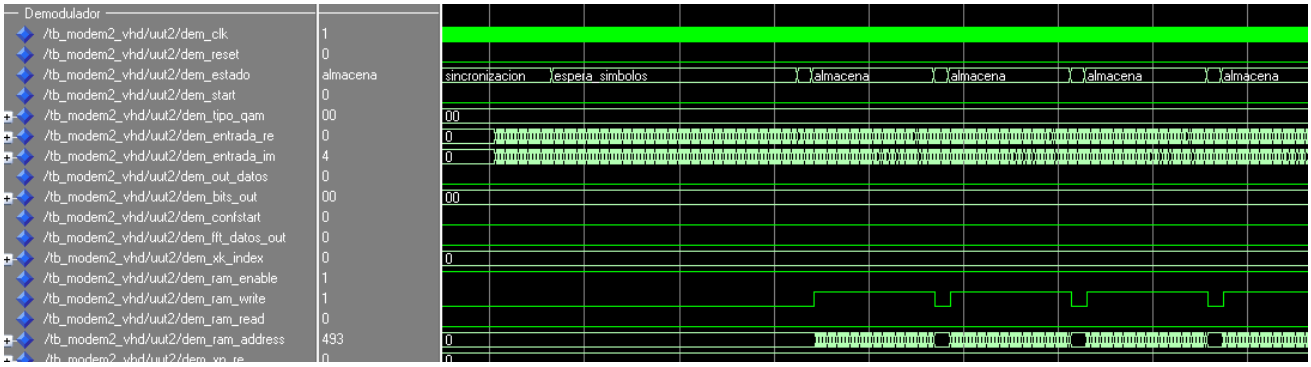


Figura 4.48: *Almacenamiento de varios símbolos OFDM*

Cuando se han almacenado N*N_OFDM datos (en este caso 6400), el demodulador comienza a calcular la FFT en bloques de N datos.

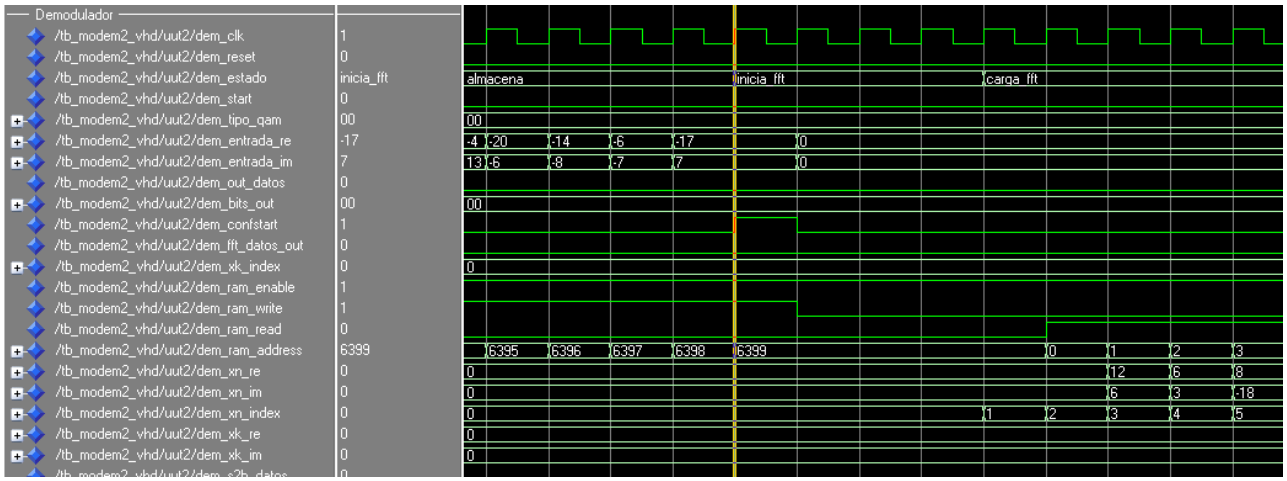


Figura 4.49: *Fin de Almacena y comienzo de Inicia_FFT*

La salida del bloque *modulo_fft* está directamente conectada al decisor, que pone los bits a la salida. En las figura 4.50 y 4.51 vemos como obtenemos en el mismo orden la secuencia de bits que teníamos en el modulador.

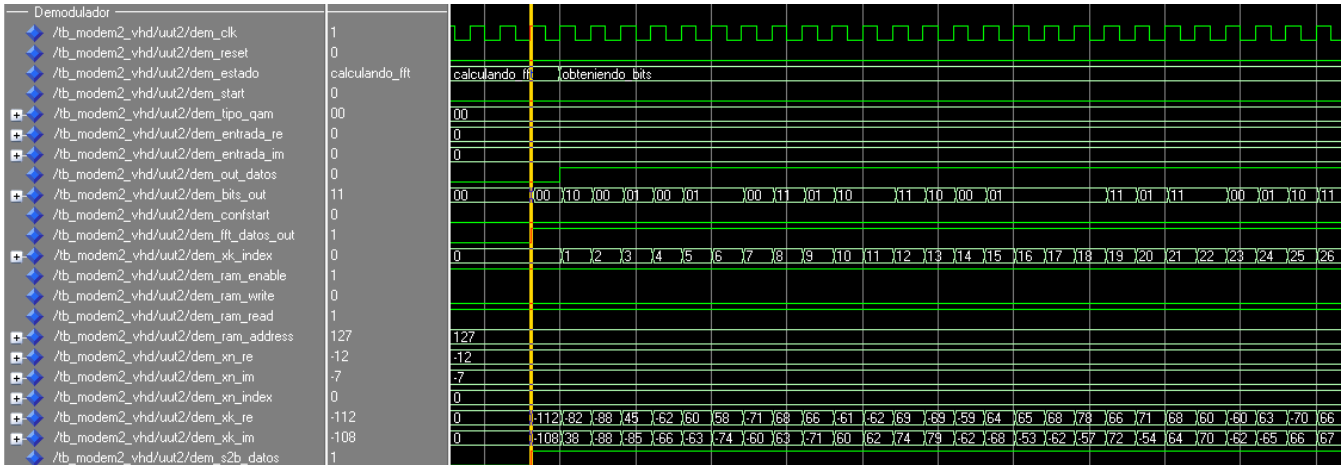


Figura 4.50: *Obtención de la primera secuencia de bits*

Cuando termina de obtener los bits del primer símbolo OFDM, vuelve a *Inicia_FFT* para continuar con el proceso. En ese momento, la salida del demodulador *dem_out_datos* vuelve a valer 0.

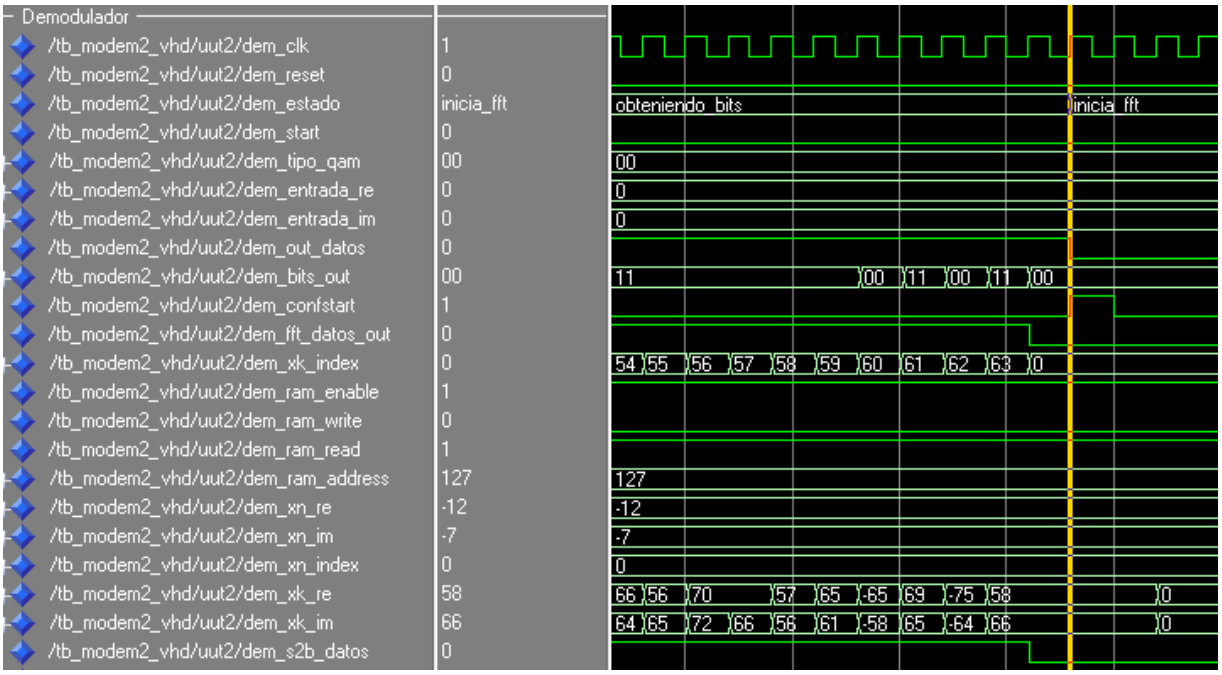


Figura 4.51: *Fin del primer símbolo e inicio del segundo*

En la figura 4.52 vemos la obtención de los bits de los cuatro primeros símbolos OFDM.

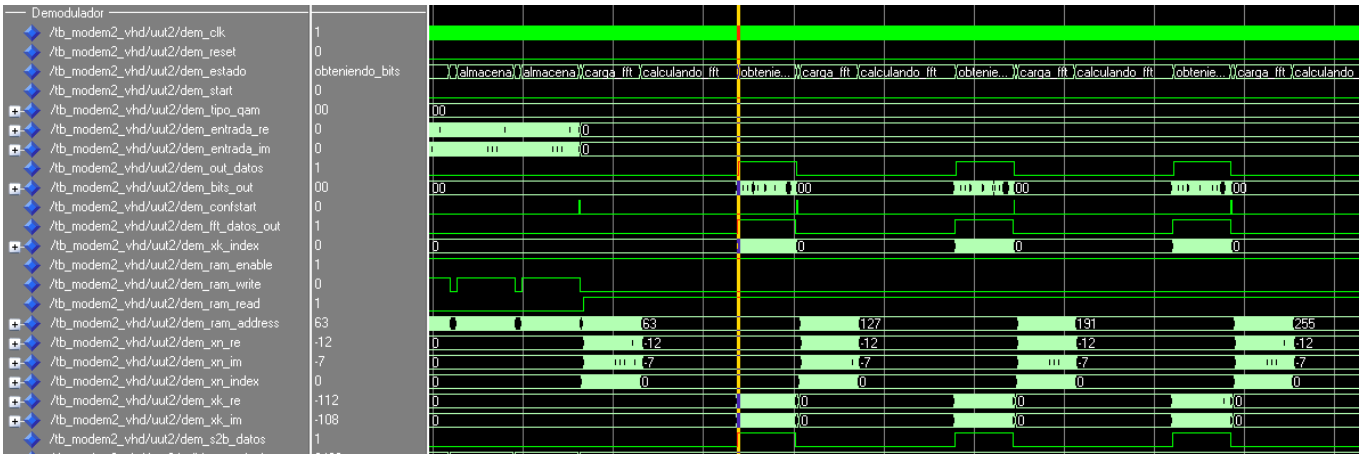


Figura 4.52: *Fin del primer símbolo e inicio del segundo*

En el último cronograma, podemos ver el proceso completo del modulador y el demodulador. En la parte superior se encuentra las señales que ve el bloque de pruebas. En la parte intermedia vemos la trama enviada y recibida. Y finalmente en la parte inferior observamos el proceso completo del demodulador.

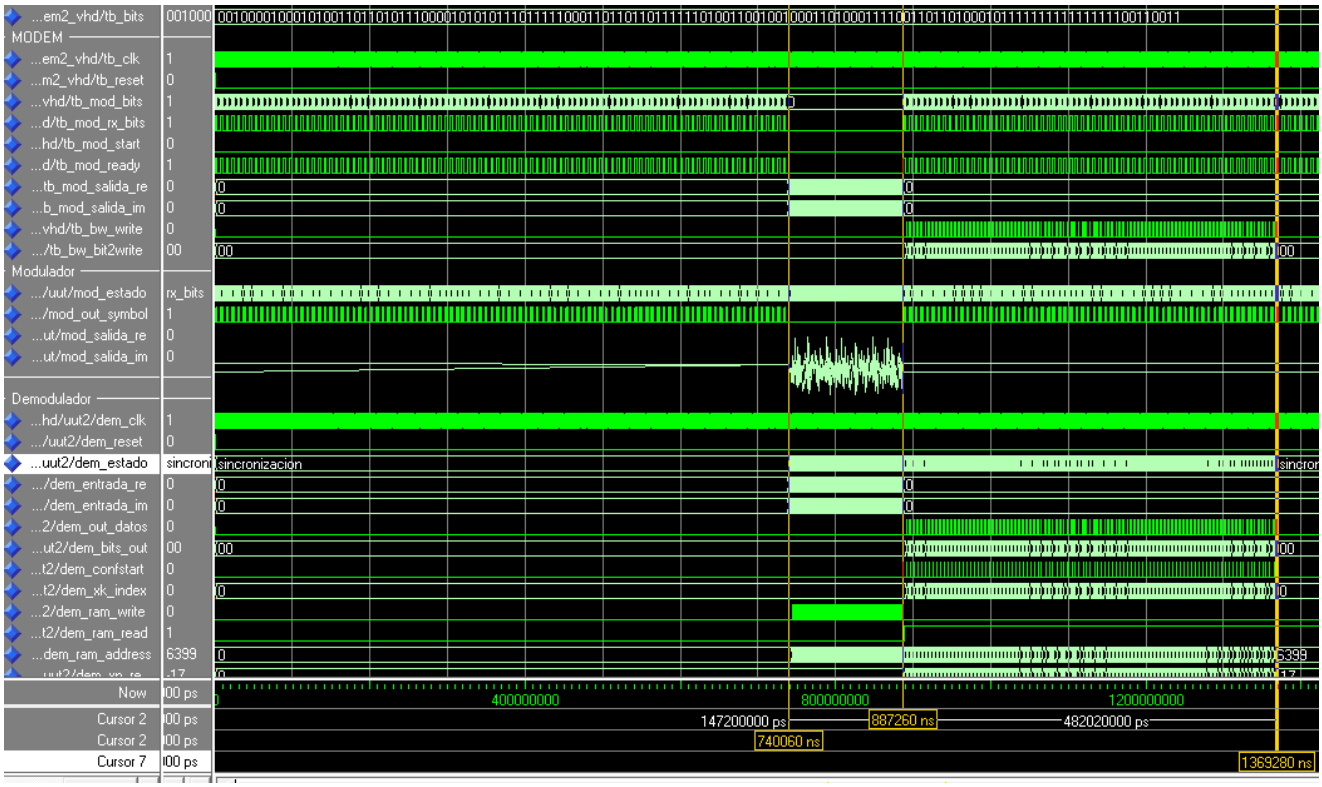


Figura 4.53: *Proceso completo del demodulador*

Para asegurarnos de que la información recibida se corresponde con la codificada en el modulador, mostramos mediante MATLAB la secuencia de bits enviada y la obtenida en el un fichero por el programa de la simulación.

En la figura 4.54 se representan los bits enviados y recibidos del primer símbolo OFDM.

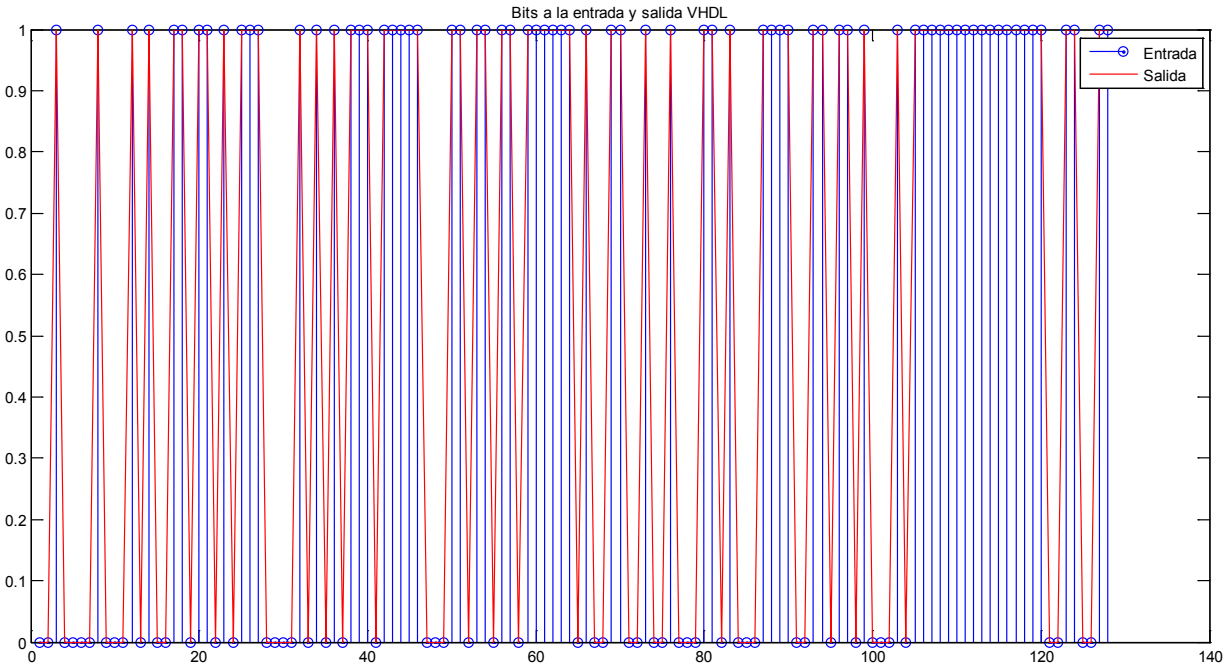


Figura 4.54: *Secuencia de bits enviados y recibidos*

5 IMPLEMENTACIÓN EN UN DISPOSITIVO SFF SDR

5.1 INTRODUCCIÓN

El objetivo principal establecido al comienzo de este proyecto era llegar a obtener resultados a nivel de hardware de una implementación en el dispositivo SFF SDR comentado en el punto 2.1. En este capítulo veremos cómo conseguir que las simulaciones de nuestro sistema OFDM explicadas en el capítulo 4, funcionen a nivel hardware en dicho dispositivo.

En el presente proyecto, se ha implementado un transmisor para comprobar el correcto funcionamiento del modulador OFDM desarrollado en el punto 4.3.1, obteniéndose resultados medibles de la señal OFDM transmitida.

Todo el desarrollo de éste punto está basado en los manuales del fabricante [5-12] y [32-34], y consta de varias fases que veremos en el punto 5.2, en las que, además del desarrollo VHDL, hay que programar el módulo de conversión de datos y el módulo de radio-frecuencia.

Una vez hecha la implementación, comentaremos los resultados obtenidos en el laboratorio de trabajo en el punto 5.3.

5.2 INTEGRACIÓN EN EL DISPOSITIVO SFF SDR

5.2.1 Integración Virtex-4

En primer lugar llevaremos a cabo la implementación del desarrollo VHDL, para lo que tendremos que utilizar las especificaciones dadas por el fabricante. En un dispositivo SFF SDR, esta implementación se lleva a cabo en una FPGA Virtex-4 (en concreto, Virtex-4 modelo XC4VSX35).

En este punto, veremos una breve descripción de las librerías proporcionadas por el fabricante para el desarrollo de programas en VHDL, desarrollaremos un módulo transmisor que genere una señal OFDM, utilizando las interfaces de las librerías y sintetizaremos y programaremos la FPGA con dicho desarrollo.

5.2.1.1 Librerías

A) Distribución

Con la instalación de las utilidades proporcionadas por Lyrtech, el usuario obtiene las carpetas de la tabla 5.1 [5] con las librerías necesarias para implementar un programa VHDL.

Carpeta	Descripción
fpga\default\netlist	Archivos Netlist precompilados para las interfaces por defecto del dispositivo.
fpga\default\src	Código VHDL fuente.
fpga\common\netlist	Archivos Netlist precompilados para las interfaces comunes del dispositivo
fpga\common\src	Archivo de constantes (.ucf) por defecto.
fpga_cores\Xilinx_comp	Archivos Netlist precompilados y archivos VHDL para los cores de Xilinx utilizados por Lyrtech.

Tabla 5.1: *Librerías VHDL del dispositivo SFF SDR*

B) Recursos utilizados

Al cargar estas librerías en el proyecto, se utilizan una parte de los recursos hardware de la FPGA, pero como vemos en la tabla 5.2 sacada de [5], apenas suponen un 15% de los recursos totales, por lo que no nos veremos limitados en este sentido.

Recursos	Usados	Accesibles	% de uso
Registros	±1696	15360	11
DCM	1	8	12
BUFG	5	32	15
RAMB16	3	192	1

Tabla 5.2: *Distribución por defecto de los recursos*

C) Interfaces proporcionadas

Desde el punto de vista del programador, al cargar todas estas librerías en el proyecto desarrollado, se implementan una serie de interfaces que permiten el desarrollo e integración del programa en el dispositivo SFF SDR. Mediante estas interfaces de entrada/salida podremos interactuar con los otros módulos del dispositivo.

En este punto, simplemente revisaremos los módulos de la figura 5.1 utilizados para llevar a cabo nuestra implementación.

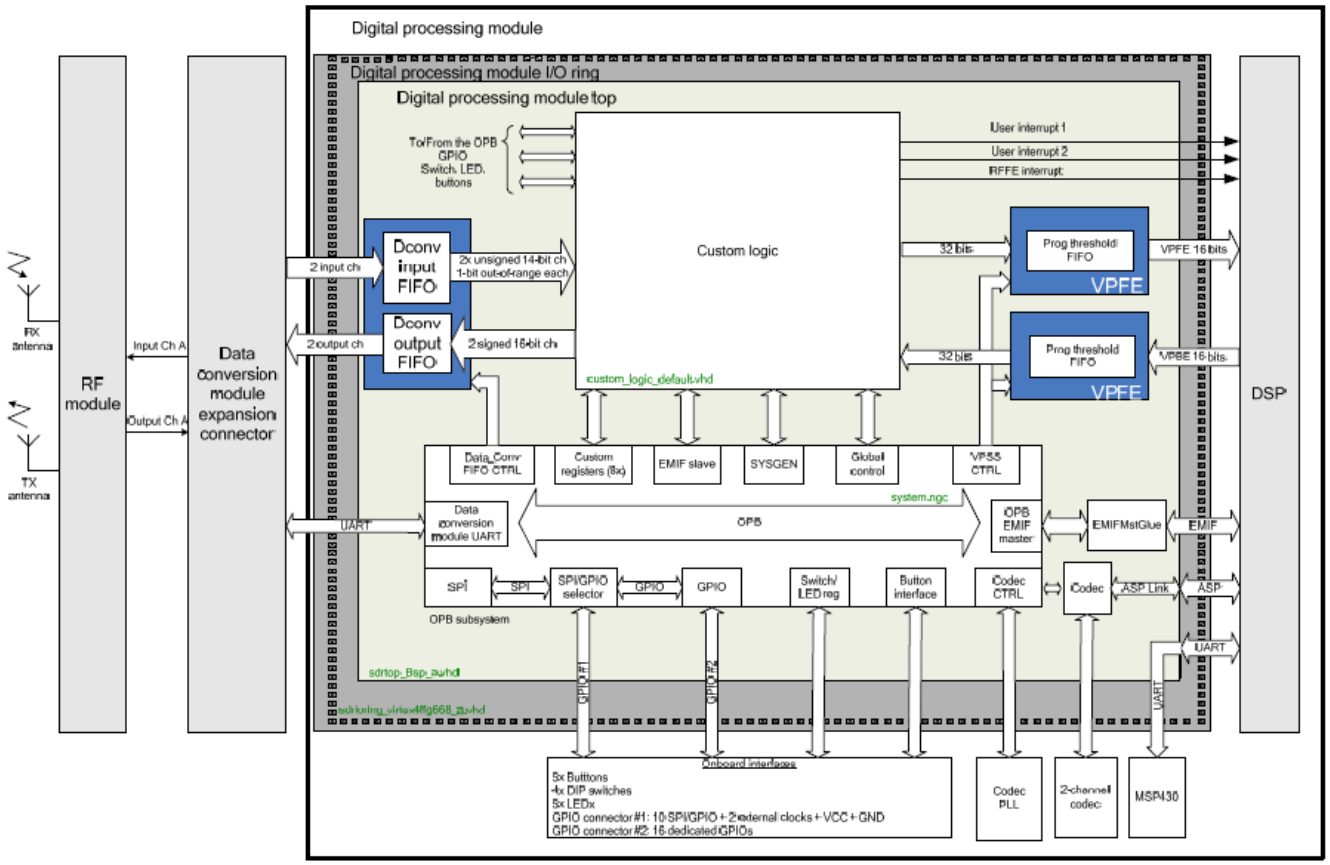


Figura 5.1: *Módulos y librerías del dispositivo SFF SDR*

- **Lista de Módulos utilizados**

  • **I/O ring**: Interfaz superior de la FPGA.
  • **FPGA top**: Interfaz que engloba a los demás bloques inferiores.
  • **Dconv Input FIFO**: Interfaz de entrada del modulo de conversión de datos.
  • **Dconv Output FIFO**: Interfaz de salida del modulo de conversión de datos.
  • **Custom logic**: Interfaz donde desarrollar el código VHDL.

5.2.1.2 Transmisor OFDM

En el punto anterior hemos visto una visión general de las interfaces proporcionadas por las librerías del dispositivo. En este punto, desarrollaremos el programa que queremos implementar y probar, utilizando las señales de las interfaces necesarias.

A) Diseño

Con el fin de probar la implementación hardware de la manera más sencilla posible, llevamos a cabo un bloque, llamado *transmisor*, con la particularidad de que funciona simplemente con la entrada de un reloj. No necesita ninguna otra señal de activación y la señal OFDM obtenida proviene de los bits de información almacenados en el módulo.

El funcionamiento del bloque *transmisor* es muy simple y sólo se encargará de sincronizar la carga de bits en el modulador del punto 4.3.1, de manera que al terminar de enviar una trama, vuelva a empezar y cargar nuevos bits que enviar.

Por último, en cuanto al diseño se ha añadido una señal de *busy* al bloque transmisor, para que en la integración en la FPGA se pueda indicar mediante el encendido y apagado de los LEDs, la transmisión de información. En la figura 5.2 vemos una representación del bloque transmisor.

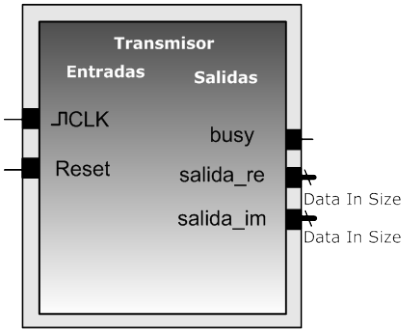


Figura 5.2: *Bloque transmisor*

B) Funcionamiento

El funcionamiento del transmisor es el representado en el diagrama de la figura 5.3. Primero resetea el modulador para asegurarse el reinicio del sistema y entra en el estado "*Cargando*". Después, mientras el modulador está listo (señal *ready* activa), el sistema de control carga los bits de memoria activando las señal *rx_bits* del modulador. Cuando el modulador comienza a transmitir datos, pasa al estado de "*Transmión*" y activa la señal de *busy*. Al terminar de transmitir comienza a cargar bits de nuevo.

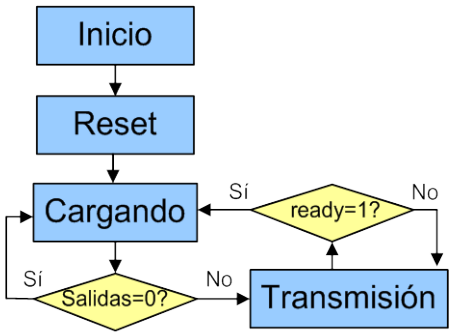


Figura 5.3: *Diagrama de estados del transmisor*

En la figura 5.4 vemos cómo queda la relación entre el transmisor y el modulador. Las señales *bits*, *rx_bits* y *ready* están conectadas a la lógica de control del transmisor.

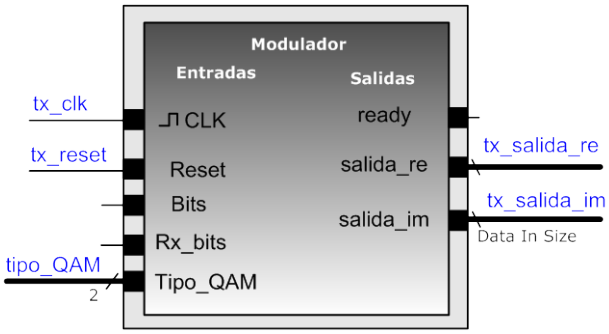


Figura 5.4: *Conexiones transmisor-modulador*

C) Simulaciones

Para probar el módulo transmisor implementamos un programa de pruebas, que simplemente escoge una frecuencia de reloj e instancia el módulo transmisor. Vemos en la figura 5.5 el resultado de la simulación del bloque transmisor.

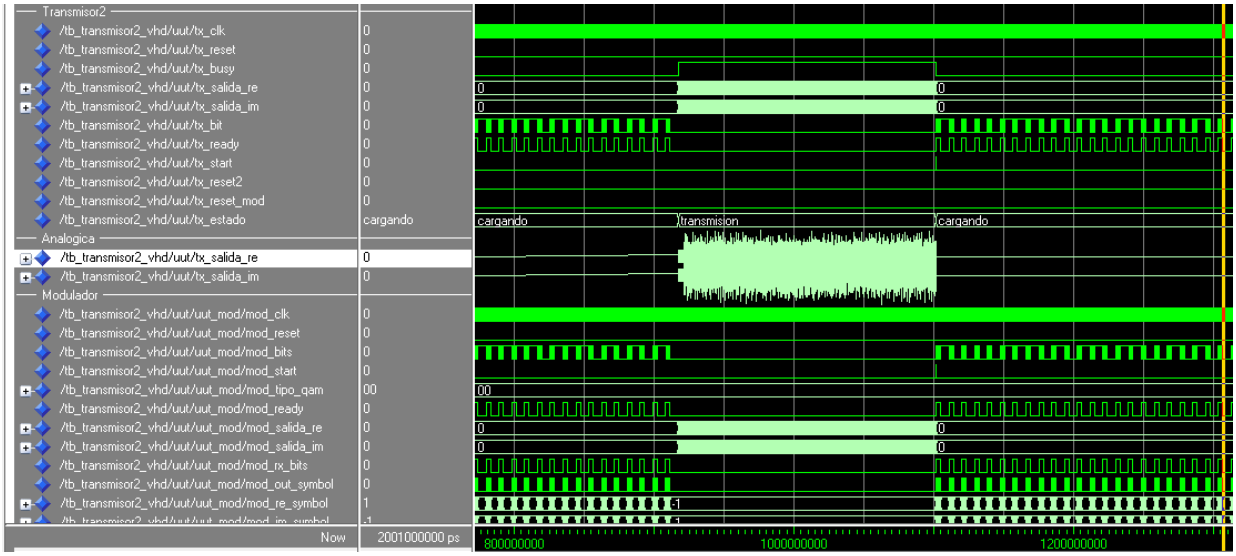


Figura 5.5: *Transmisión de una trama de datos*

D) Relación con las interfaces SFF SDR

Además de la instanciación del componente *transmisor* utilizaremos algunas de las señales que nos proporcionan las interfaces explicadas en el punto 5.2.1.1. El resto de interfaces de la figura 5.1 las dejaremos asignadas a su valor por defecto.

Como ya hemos comentado, el transmisor sólo necesita como entrada una señal de reloj que obtendremos de las señales de reloj de entrada a *Custom_Logic*. Utilizaremos la señal del reloj principal de la FPGA, *i_FpgaClk_p,* mediante un contador que nos divida la frecuencia por un número entero.

La decisión en la elección de la frecuencia de trabajo la tomaremos en el punto 5.3 ya que a priori, el único requisito que limita el ancho de banda de la señal es el del filtro de entrada al módulo RF (65 MHz) y que cumplimos.

En la tabla 5.3 podemos ver las señales de entrada y salida del bloque *Custom_Logic* utilizadas.

Nombre del puerto	Dirección	Descripción
i_FpgaClk_p	Entrada	Reloj principal de la FPGA (37.5 MHz)
o_ClkCstm2Dac_p	Salida	Reloj de entrada del modulo de conversión de datos (Máximo 125 MHz).
ov16_DacDataChA_p	Salida	Bus de salida al canal A (con signo)
ov16_DacDataChB_p	Salida	Bus de salida al canal B (con signo)
o_DacWr_p	Salida	Señal de escritura de la salida FIFO

Tabla 5.3: *Señales de entrada/salida del Custom_Logic utilizadas*

La elección de la frecuencia de reloj del DAC va estrechamente ligada a la configuración del módulo de conversión de datos, el cual, se programa a través de la API de Lyrtech a través del Code Composer Studio como veremos en el punto 5.2.2.

Por lo tanto, en este apartado nos limitaremos a indicar que deberemos escoger una frecuencia de trabajo para el DAC acorde a la configuración elegida y que no sobrepase los 125 MHz como vemos en la tabla 5.3. Y la elección de la frecuencia del reloj la realizaremos en el punto 5.3.

Las dos señales *ov16_DacDataChA_p* y *ov16_DacDataChB_p* las conectamos a la salida real y compleja del transmisor amplificándolas multiplicando por una constante. La señal *o_DacWr_p* la mantenemos siempre a nivel alto, para que la salida del *transmisor* esté continuamente entrando en el DAC.

5.2.1.3 Síntesis y programación

Una vez llegados a este punto, podemos sintetizar y programar el código VHDL para obtener un archivo *bit* programable en una FPGA Virtex-4.

La síntesis de todos los módulos y librerías la lleva a cabo el programa Xilinx ISE mediante la herramienta XST (Xilinx Synthesis Tools). Mediante esta herramienta de síntesis se genera a partir de código VHDL una netlist (archivo que describe la conectividad de un diseño electrónico) en tres etapas: Análisis, Compilación y Mapeo.

La creación del archivo *bit*, también se hace mediante una herramienta del propio Xilinx ISE: BitGen, que genera el archivo programable en la FPGA.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	5,273	30,720	17%	
Number of 4 input LUTs	6,157	30,720	20%	
Logic Distribution				
Number of occupied Slices	4,424	15,360	28%	
Number of Slices containing only related logic	4,424	4,424	100%	
Number of Slices containing unrelated logic	0	4,424	0%	
Total Number of 4 input LUTs	6,921	30,720	22%	
Number used as logic	6,157			
Number used as a route-thru	188			
Number used for 32x1 RAMs	12			
Number used as 16x1 RAMs	16			
Number used as Shift registers	548			
Number of bonded IOBs	212	448	47%	
Number of BUFG/BUFGCTRLs	9	32	28%	
Number used as BUFGs	8			
Number used as BUFGCTRLs	1			
Number of FIFO16/RAMB16s	29	192	15%	
Number used as FIFO16s	0			
Number used as RAMB16s	29			
Number of DSP48s	20	192	10%	
Number of DCM_ADVs	2	8	25%	
Number of RPM macros	3			
Total equivalent gate count for design	2,029,248			
Additional JTAG gate count for IOBs	10,176			

Figura 5.6: *Señales de salida del Custom_Logic utilizadas*

En la figura 5.6 podemos observar una captura de pantalla con los resultados obtenidos en la sintetización llevada a cabo, de donde podemos sacar una serie de apreciaciones:

- Todos los registros utilizados contienen un comportamiento lógico
- La estructura de datos (LUT) está ocupada al 22%
- La mayor utilización se encuentra en los bloques de entrada salida (IOB) y llega al 47%

En resumen, vemos como en general, se han empleado menos de la mitad de los recursos disponibles, con lo que la utilización se ha conseguido de una manera eficiente, lo que beneficiará al comportamiento y rapidez del sistema.

5.2.2 Configuración de los módulos de conversión de datos y de radio-frecuencia

La configuración de los módulos de conversión de datos y de radio-frecuencia se lleva a cabo programando el DSP como ya se explicó en el punto 2.1.

Utilizando el entorno de desarrollo CCS, generamos un archivo *out* que se carga en el DSP y configura los módulos. Para ello, el fabricante proporciona una API [32] con los métodos necesarios.

En nuestro caso, sólo programaremos el DAC del módulo de conversión de datos, dejando que el módulo de RF module las señales en fase y cuadratura a la banda por defecto. En este punto por tanto, hacemos una breve descripción de los métodos de la API utilizados para configurar el DAC del dispositivo SFF SDR.

En primer lugar, hay que inicializar la FPGA mediante los métodos *INIT_Chip()* y *FPGA_Init()*. La inicialización del módulo de conversión de datos, en cambio, necesita la llamada a tres métodos: *conv_mod_InitPrologue(), conv_mod_init()* y *conv_mod_InitEpilogue().*

Una vez hecho esto, para configurar el DAC, sólo tendremos que llamar a los métodos *conv_mod_enableDAC()* y *conv_mod_SetDualChanDACMode()*. El primero activa el DAC del módulo de conversión de datos y el segundo configura que las dos entradas se interpolen de forma independiente. Las posibles interpolaciones son x1, x2, x4 y x8, y se puede usar una corrección sin(x)/x en la interpolación.

En la API se suministran a su vez varios métodos para configurar el reloj del DAC. Si no utilizamos ninguno de los métodos, el DAC utilizará su máxima frecuencia de trabajo (125 MHz), lo que nos resultará bastante útil en las pruebas realizadas (5.3).

Por último notar que en el archivo C donde hemos utilizado estos métodos, tendremos que incluir las librerías del dispostivo "*davincidef.h*", "*FPGA.h*" y "*ConversionModule.h*".

5.3 RESULTADOS OSCILOSCOPIO

En este punto pasamos a mostrar y analizar los resultados obtenidos en el laboratorio. Utilizando conectores SMA como vemos en el entorno de trabajo del laboratorio de la figura 5.7, podemos ver la salida del conversor de datos y del módulo de RF en el osciloscopio.

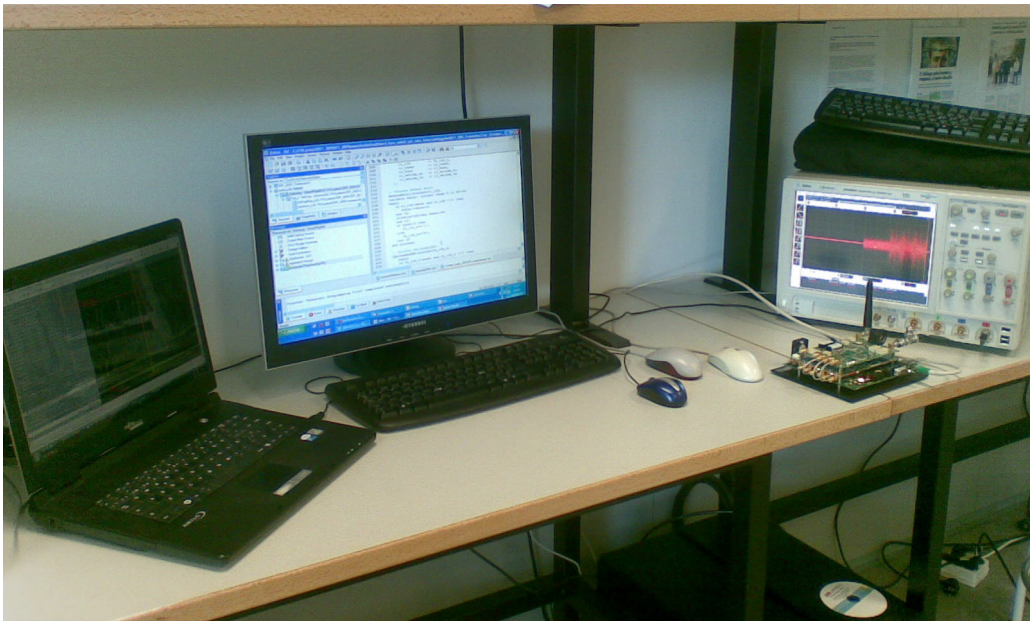


Figura 5.7: *Entorno de trabajo en el laboratorio*

En primer lugar, vemos unas características de decisiones tomadas en el programa implementado (5.3.1), y en los puntos 5.3.2 y 5.3.3 vemos los resultados obtenidos, teniendo en cuenta que a la salida del conversor de datos tenemos las señales de fase y cuadratura separadas, y a la salida del módulo RF tenemos la señal OFDM modulada en banda.

5.3.1 Programa de pruebas

Cargamos los ficheros *bit* y *out* en el dispositivo con las siguientes características:

- Frecuencia de la señal 9.375 MHz (Reloj FPGA dividido por cuatro).
- Frecuencia de trabajo del DAC 125 MHz.
- Interpolacion x8
- Banda de radio-frecuencia: 543 MHz

La decisión de la frecuencia de trabajo viene dada por el hecho de querer reducir la frecuencia principal del reloj de la FPGA. Reduciendo a cuatro veces más lenta esta frecuencia y junto con la interpolación x8 se han obtenido unos buenos resultados.

5.3.2 Salida del módulo de conversión de datos

En primer lugar observamos la salida del **canal A** del conversor de datos que se corresponde con la parte de la señal OFDM real. Los resultados del otro canal, son similares, por lo que evitamos mostrarlos dos veces. En la figura 5.8 vemos las tramas OFDM que se envían y en la figura 5.9 vemos una de esas tramas.

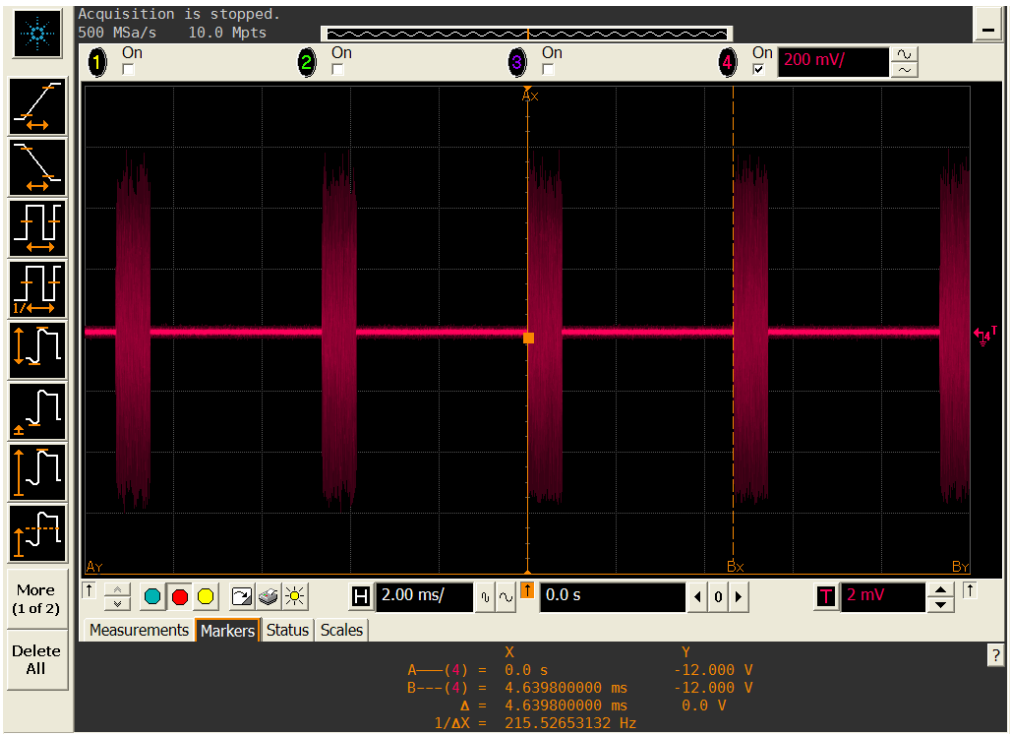


Figura 5.8: *Varias tramas OFDM a la salida del conversor de datos*

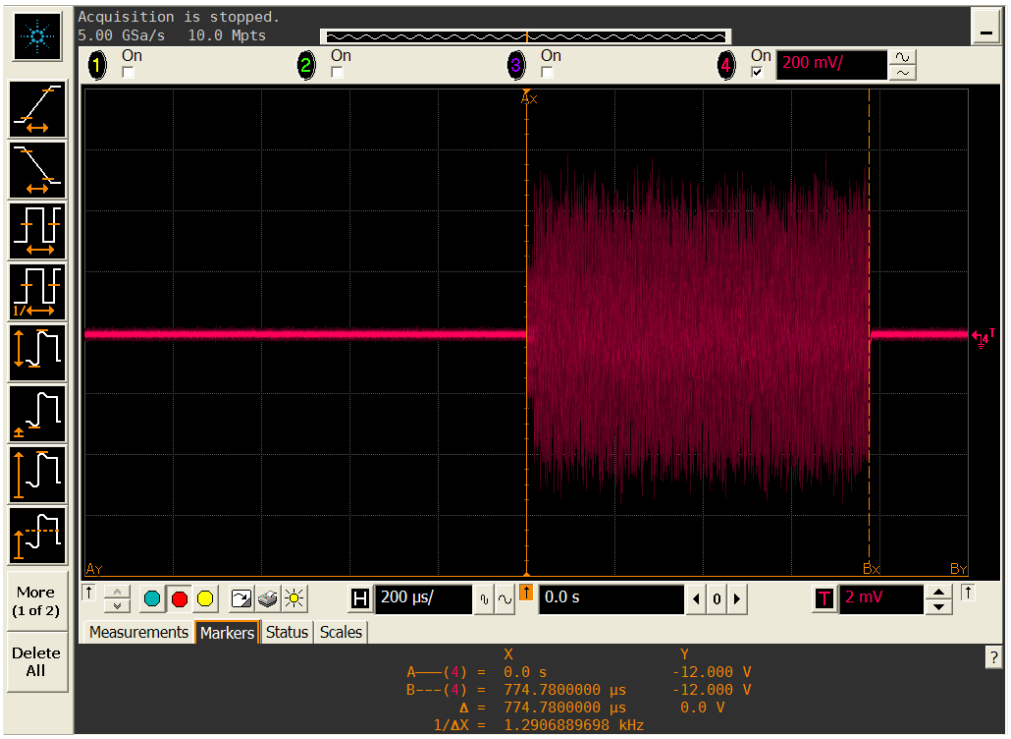


Figura 5.9: *Trama OFDM a la salida del conversor de datos*

Vemos como, tanto los tiempos entre tramas, como la duración de las mismas, coinciden con la simulación del transmisor hecha a frecuencia 9.375 MHz, de la figura 5.10.

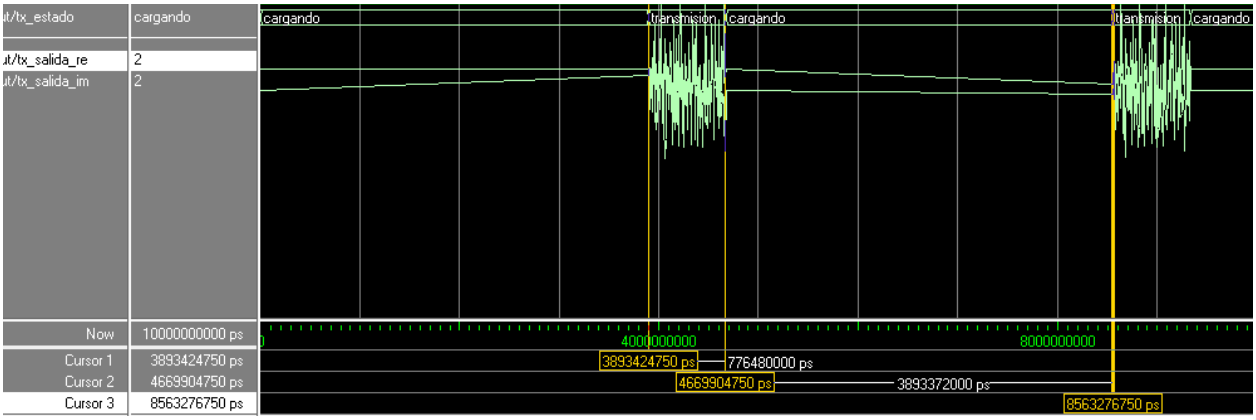


Figura 5.10: *Tiempos en la simulación del transmisor a 9.375 MHz*

Al igual que ocurría en las simulaciones, la amplitud de los preámbulos es menor que la de los símbolos OFDM, lo que nos ayuda a identificar ambas partes en la señal.

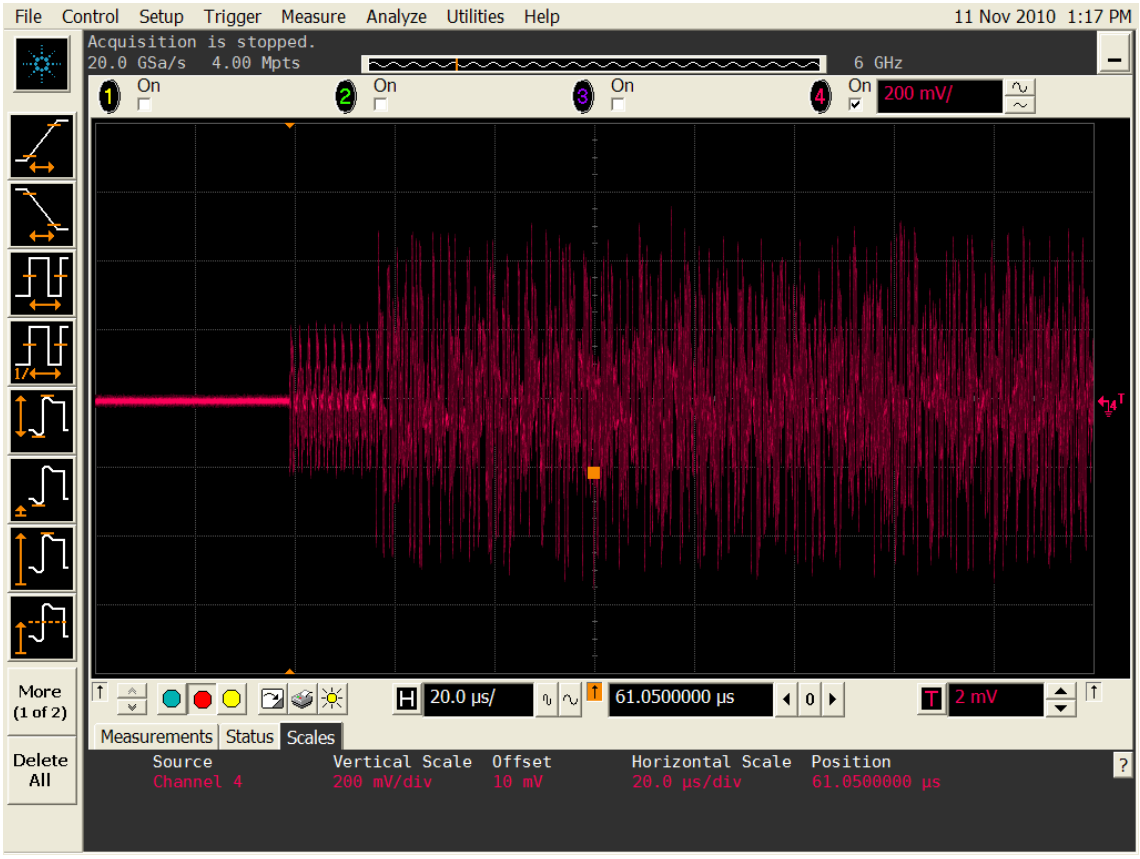


Figura 5.11: *Trama OFDM a la salida del conversor de datos ampliada*

Si en lugar de interpolación x8, no interpolamos (interpolación x1), vemos como la frecuencia de trabajo es de 9.375 MHz.

Figura 5.12: *Frecuencia de datos a 9.375 MHz*

Comparamos, mirando los preámbulos, los cuatro tipos de interpolaciones comentadas en el punto 5.2.2, para comprobar como mejora la señal según aumentamos en valor de la interpolación.

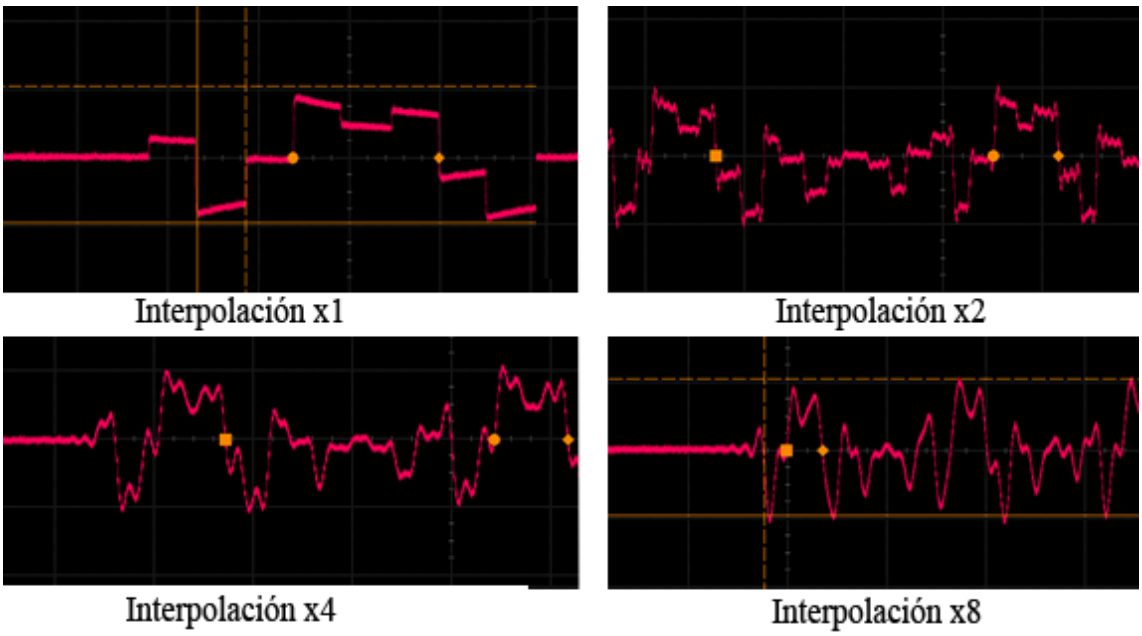


Figura 5.13: *Preámbulos con diferentes interpolaciones*

Vemos como la interpolación x8 es la que mejor obtiene una señal analógica. Si superponemos la salida de dicha interpolación, sobre la figura 5.14 que obtuvimos en el capítulo de teoría, vemos como coinciden.

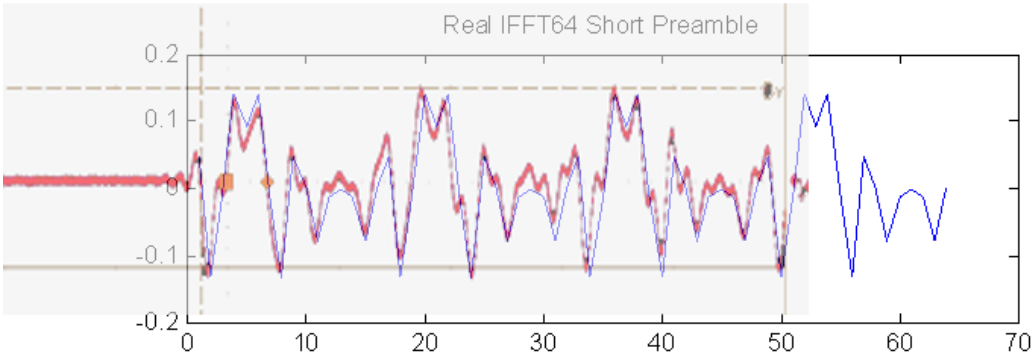


Figura 5.14: *Superposición de interpolación x8 en los preámbulos*

Si miramos la FFT de la señal que obtenemos en el canal A (señal amarilla en la figura 5.15), observamos como tenemos el espectro de una señal OFDM con réplicas en 125 MHz (cursor B), debido a la frecuencia de trabajo del módulo de conversión de datos.

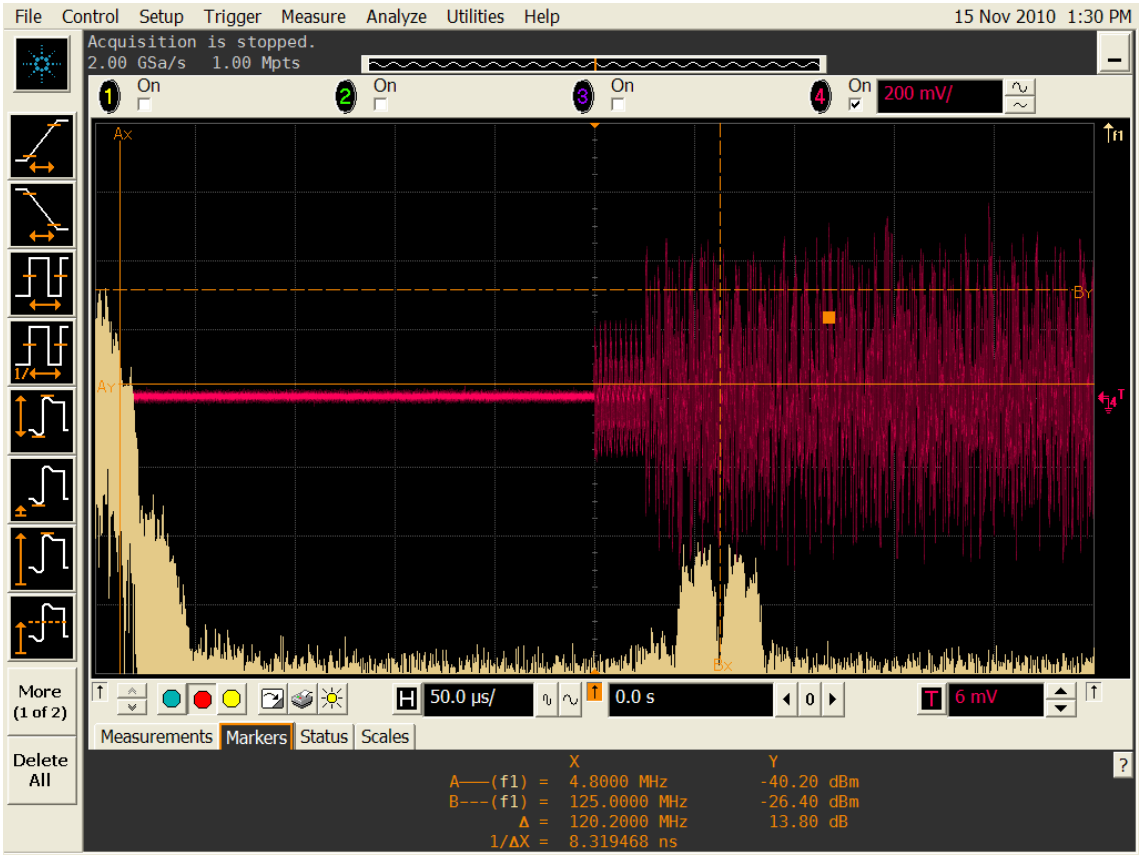


Figura 5.15: *FFT de la señal OFDM real (canal A)*

Aumentando la resolución de la FFT, vemos en la figura 5.16 un lóbulo principal de espectro plano de ancho 4.7 MHz que es $F_s/2$. También podemos observar un segundo escalón, proveniente de los lóbulos secundarios, de ancho de banda de 7.8 MHz que permanece debido a al ancho de banda del filtro interpolador (125 MHz/8/2=7.8 MHz), pero con una amplitud considerablemente menor.

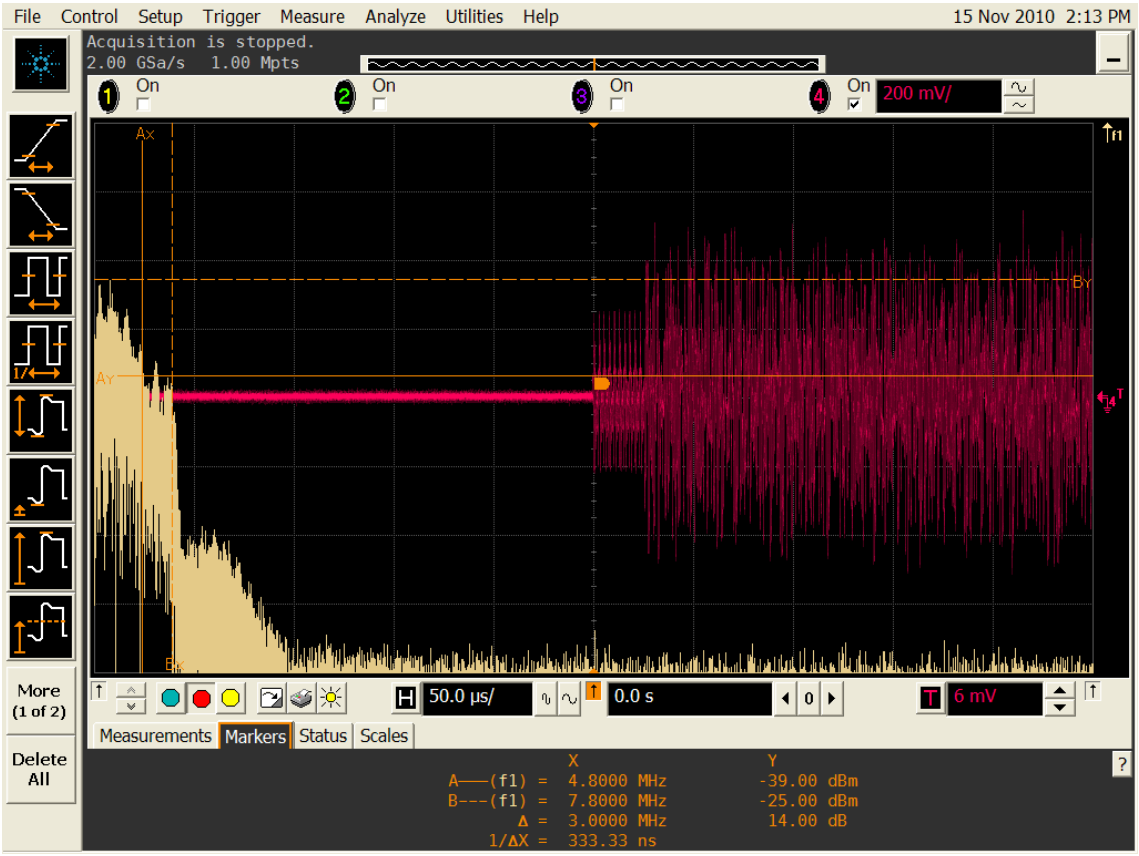


Figura 5.16: *BW de Fs/2 en la FFT de la señal OFDM real (canal A)*

5.3.3 Salida del módulo de radio-frecuencia

Para poder visualizar la salida de la señal modulada en banda, conectamos la salida del módulo de radio frecuencia al osciloscopio.

En la figura 5.17 podemos ver como tenemos una señal modulada en banda. Y aumentando la resolución en las figuras 5.18 y 5.19 observamos como la frecuencia de la banda de transmisión es de 543 MHz. Esta frecuencia la selecciona el módulo RF por defecto.

Figura 5.17: *Señal OFDM modulada en RF I*

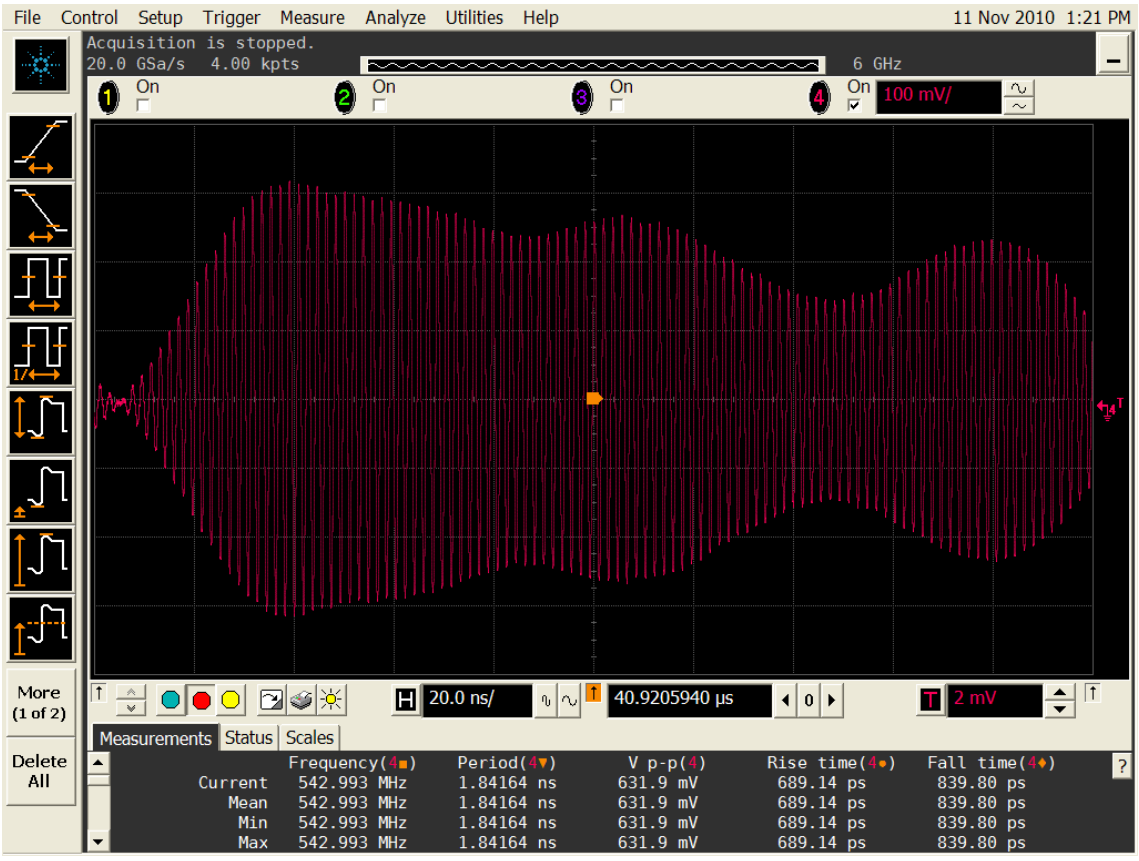


Figura 5.18: *Señal OFDM modulada en RF II*

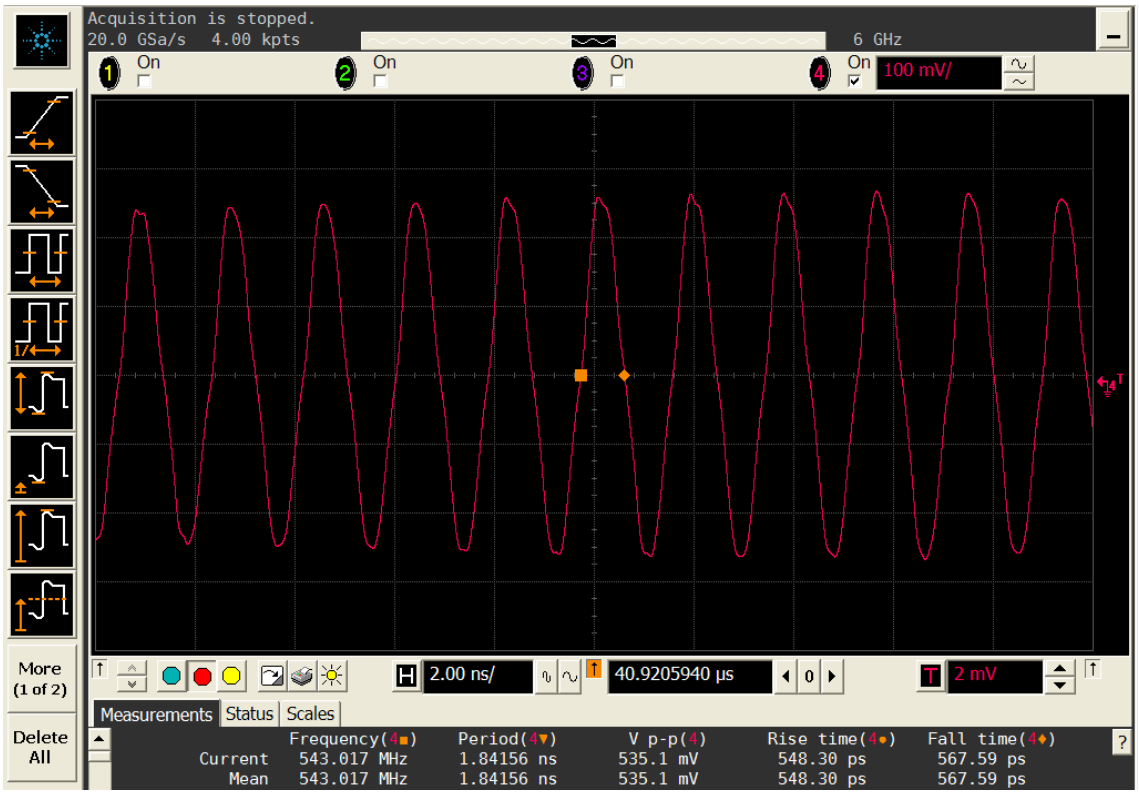


Figura 5.19: *Señal OFDM modulada en RF III (543 MHz)*

Representando la FFT (señal amarilla) en un amplio rango de frecuencias, vemos como el lóbulo principal está en la banda de RF.

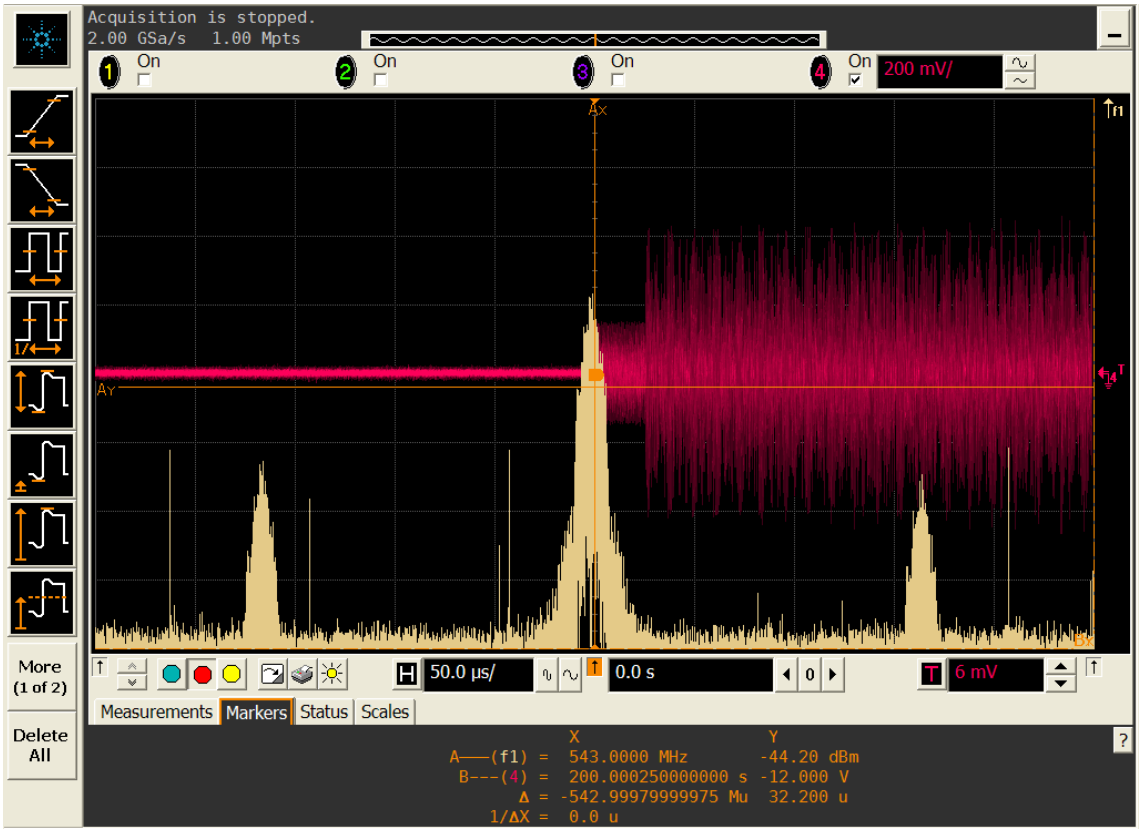


Figura 5.20: *FFT de la señal OFDM modulada en RF (543 MHz)*

Delimitando el ancho del lóbulo principal (figura 5.21) y ampliando la resolución del espectro (figura 5.22), vemos como la FFT de la señal RF es un espectro plano de ancho de banda 9.375 MHz. Este valor coincide con lo que vimos en el apartado de teoría 3.2.2, es decir, el ancho de banda de la señal transmitida es igual a la frecuencia de trabajo del transmisor.

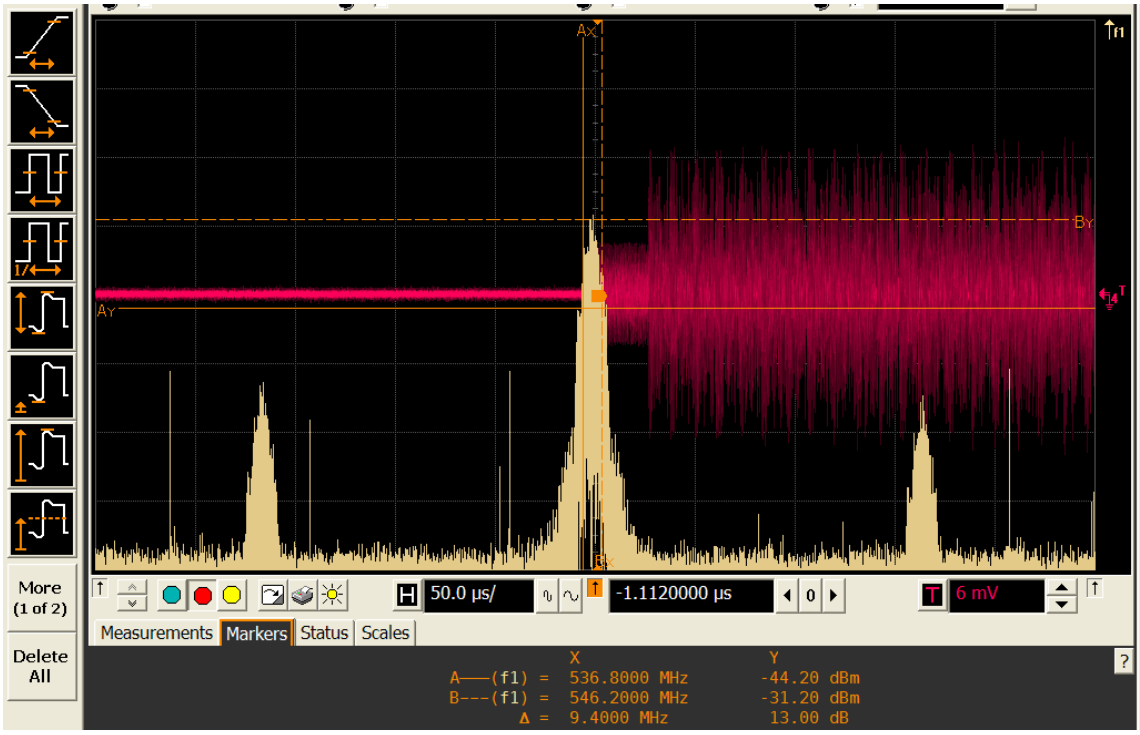


Figura 5.21: *BW de la señal OFDM modulada en RF I*

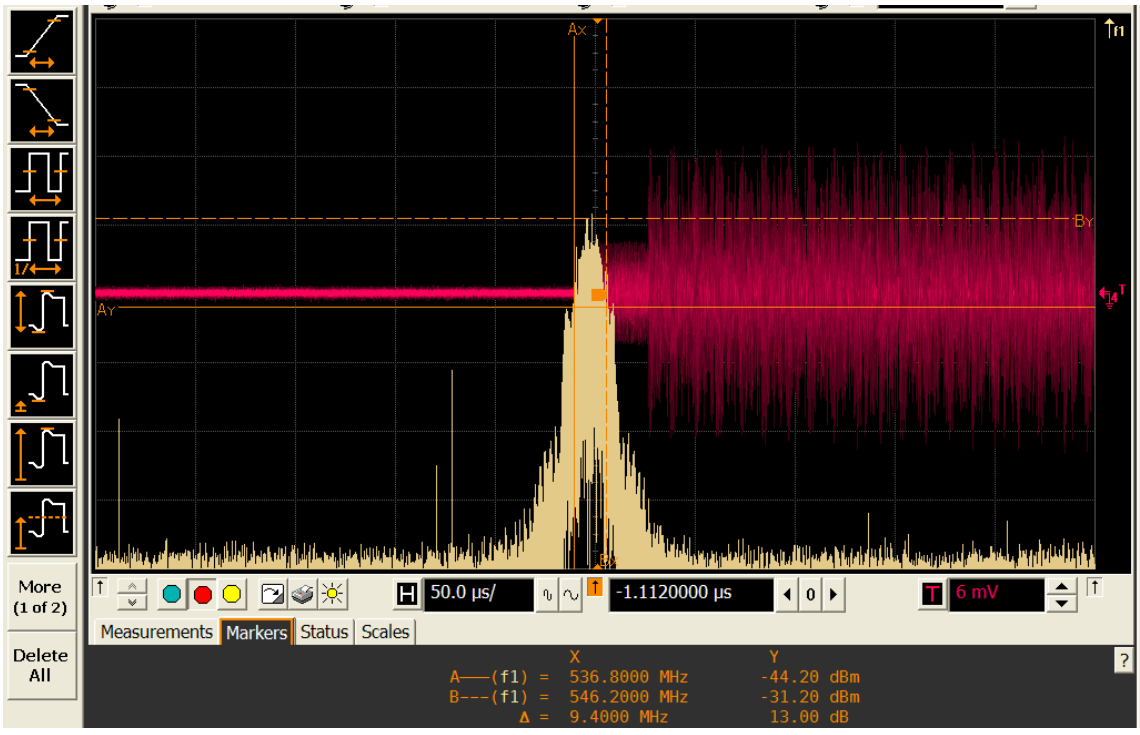


Figura 5.22: *BW de la señal OFDM modulada en RF II*

5.4 ANÁLISIS DE RESULTADOS

Analizando los resultados obtenidos en el punto anterior podemos sacar una serie de conclusiones sobre las cualidades de la señal transmitida, que pasamos a enumerar:

- Se ha conseguido configurar y programar el dispositivo SFF SDR obteniéndose una señal de transmisión, modulada en una banda de RF.

- Los resultados obtenidos a la salida del "módulo de conversión de datos" concuerdan, en fase y cuadratura, con el esquema básico de un modulador en un sistema de comunicaciones OFDM.

- La conversión digital-analógica mantiene las características de la señal digital obtenida de la modulación.

- Respecto al espectro de transmisión, perdemos una cierta eficiencia de ancho de banda debida al filtro interpolador, que deja pasar parte de los lóbulos secundarios. Sin embargo, debido a las características del DAC, un ajuste mayor del BW suponía comenzar a perder información en la señal analógica. Por tanto, se considera que los resultados obtenidos, son los que han proporcionado un mejor comportamiento.

6 HISTORIA DEL PROYECTO

El desarrollo del presente proyecto se ha realizado en el periodo de tiempo comprendido entre Septiembre de 2009 a Septiembre de 2010.

Durante el año de trabajo, el proyecto se ha desarrollado en diferentes fases que se describen en este capítulo. De cada fase, identificaremos los objetivos establecidos, los problemas encontrados para la consecución de los mismos, y las soluciones y decisiones tomadas.

6.1 FASES DEL PROYECTO

6.1.1 Familiarización con el entorno de trabajo

A) Descripción general

Uno de los objetivos del presente proyecto era llevar a cabo la implementación en un dispositivo SFF SDR. Para lo cual, ha sido necesario llevar a cabo una familiarización con las herramientas provistas por el fabricante.

En esta fase se instalaron los programas necesarios para configurar el dispositivo y cargar los ficheros. Así como se revisaron las características principales del lenguaje de programación VHDL.

Esta fase se ha alargado a prácticamente la totalidad de la duración del proyecto, ya que al ser la primera vez que se implementaba un sistema de comunicaciones en este dispositivo mediante el camino empleado, suponía el desconocimiento a priori de los métodos y procedimientos necesarios.

B) Problemas

La instalación de los programas no resultó tan simple como cabría esperar. Las licencias que se tenían de los programas Xilinx ISE o Code Composer Studio, pertenecían a versiones de programas de hace unos años, lo que ha supuesto una complicación a la hora de buscar información complementaria a los manuales del fabricante.

Además, los programas se necesitaban en las versiones que las librerías del dispositivo SFF SDR preveían, por lo que ha habido que gestionar la renovación de las licencias.

Por otro lado, en la instalación del entorno de desarrollo del código VHDL se encontró algún problema en la compilación de librerías que permitieran la simulación del código.

C) Resultados

Una vez activadas las licencias, se consiguió que el software instalado detectara tanto el dispositivo como las librerías del mismo. Por lo que se pudo probar mediante la carga de los programas de prueba proporcionados por el fabricante.

También se consiguió instalar el software de simulación con lo que se facilitó el desarrollo del código en paralelo con la configuración del dispositivo SFF SDR.

6.1.2 Definición de requisitos

A) Descripción general

Partiendo de los objetivos del proyecto, en esta fase se definieron los requisitos necesarios que el sistema de comunicaciones debía de cumplir.

B) Problemas

La definición de requisitos se ha visto condicionada al hecho de que este proyecto suponga el comienzo de una línea de trabajo para futuras implementaciones. Por eso, el objetivo primordial era la obtención de un resultado medible, es decir, de conseguir transmitir una señal OFDM. Y la configuración y programación del dispositivo ha supuesto un gran esfuerzo debido a la complejidad del mismo y a la ausencia de soporte del fabricante.

C) Resultados

La consecuencia de este problema se refleja en que la mayor profundidad de conceptos se ha alcanzado en la parte de desarrollo y simulaciones del sistema completo; ya que la parte de integración en el dispositivo SFF SDR ha supuesto un trabajo más técnico.

6.1.3 Implementación del sistema OFDM

A) Descripción general

Ésta ha sido la fase que más desarrollo ha tenido del proyecto. En ella, se ha desarrollado el código VHDL que genera el sistema OFDM. El desarrollo ha sido incremental, tanto en el modulador como en el demodulador, es decir, primero se implementó un modulador que transmitía un símbolo OFDM y luego el demodulador que obtenía los bits. A partir de ese momento, se fueron incorporando partes más complejas: sincronización, prefijos cíclicos, N símbolos OFDM, etc.

En esta parte también se han desarrollado en VHDL programas de pruebas que evaluaran el comportamiento de los bloques implementados mediante el uso de simulaciones.

B) Problemas

Los principales problemas en el desarrollo de los bloque que formaban cada parte del sistema (modulador y demodulador), se han encontrado en el sincronismo, ya que los sistemas desarrollados en VHDL tienen que estar preparados para actuar exactamente en el instante de tiempo necesario. Esto ha supuesto, que antes de implementar nada, había que tener muy claro las señales de entrada y salida, las conexiones entre bloques y la temporización de cada bloque.

Asimismo, la simplicidad del lenguaje VHDL ha supuesto un inconveniente a la hora de implementar operaciones de cierta complejidad como es el cálculo de la autocorrelación perteneciente a la sincronización entre transmisor y receptor.

C) Resultados

Para llevar a cabo la implementación se han valorado todas las opciones que proporciona el lenguaje VHDL (diagramas de estado, definición de funciones, IP cores, etc.), escogiéndose para cada problema concreto, aquella que proporciona una mayor simplicidad y claridad.

El uso de las simulaciones ha permitido depurar el comportamiento de cada bloque individualmente, así como, la coordinación entre bloques, consiguiéndose en todos los casos el funcionamiento deseado.

6.1.4 Integración y configuración del dispositivo SFF SDR

A) Descripción general

En esta fase del desarrollo, se ha implementado el transmisor del sistema de comunicaciones en el dispositivo SFF SDR. Para ello, se configuraron los módulos de conversión de datos y de radio-frecuencia mediante la programación del módulo DSP, y la integración del código VHDL desarrollado en las librerías del dispositivo.

B) Problemas

Sin lugar a dudas, ésta ha sido la fase más complicada de superar debido a la poca información de ciertos aspectos del dispositivo dados por el fabricante (como la escasa información de los métodos de la API para el DSP y su necesidad en el funcionamiento del sistema, la no especificación de la estructura completa de las librerías VHDL y las conexiones entre las interfaces, etc.).

Además, el proceso se ha dificultado aún más debido a que el fabricante no proveía ninguna interfaz de depuración, con lo que la única información disponible tras cargar una configuración, provenía del encendido o apagado de los LEDs o de las mediciones tomadas en el laboratorio.

C) Resultados

A parte de que durante todo el tiempo del proyecto, no se ha dejado de aprender del entorno de desarrollo, aproximadamente los últimos 3 meses, se han dedicado casi exclusivamente a la configuración y la carga del programa en el dispositivo.

Durante este tiempo se fue avanzando en los problemas y posibles soluciones a tomar para conseguir que el dispositivo funcionara. La obtención de estas posibles soluciones se buscaron tanto en foros especializados como dirigiéndose al fabricante.

Al final, se consiguió medir la señal a la salida del dispositivo por lo que se pudieron comenzar a probar diferentes configuraciones para la fase de pruebas.

6.1.5 Pruebas y medición de resultados

A) Descripción general

Una vez configurado el dispositivo, en esta fase se llevaron a cabo pruebas modificando ciertos parámetros del dispositivo, observando la respuesta del mismo mediante la instrumentación requerida en un laboratorio de comunicaciones.

B) Problemas

El principal problema en esta parte ha sido una consecuencia de la configuración del módulo de conversión de datos, ya que hemos tenido que buscar un compromiso entre la calidad de la señal analógica y la eficiencia espectral de la misma.

C) Resultados

Al final, los resultados obtenidos cumplen las expectativas puestas en el dispositivo ya que configurando con diferentes parámetros (frecuencia de trabajo, interpolación, etc.) se ha obtenido la mejor respuesta posible.

Además, la utilización del un osciloscopio como el “Infinium Oscilloscope” de Agilent, ha resultado de gran utilidad en la captura de resultados.

6.1.6 Documentación

A) Descripción general

Esta fase consiste en la documentación del trabajo realizado y la realización de esta memoria.

6.2 OPINIÓN PERSONAL

La posibilidad de llevar a cabo un trabajo de implementación me ha permitido aumentar mis conocimientos de la familia de dispositivos con procesamiento digital de señales existente. Además, el haber profundizado en el dispositivo utilizado, me ha permitido lograr su correcto funcionamiento, demostrando así, la viabilidad del camino tomado.

El hecho de que este proyecto sea el primero de otros ya en marcha, sobre el mismo dispositivo, recompensa el tiempo empleado, ya que la gran parte del estudio y pruebas realizadas tendrán un aprovechamiento en los futuros proyectos.

Por otro lado, ha supuesto una satisfacción haber podido aplicar conceptos de diferentes campos estudiados durante la carrera (electrónica analógica y digital, comunicaciones digitales, sistemas de transmisión, etc.) en la consecución del proyecto.

En cuanto a experiencia personal, el presente proyecto, me ha concedido la oportunidad de profundizar en una forma de modulación con una gran presencia en las comunicaciones de hoy en día, así como expectativa en las futuras, como es OFDM.

7 CONCLUSIONES

El trabajo desarrollado en el presente proyecto buscaba iniciar una línea de trabajo en la que poder profundizar en las mejoras de los sistemas de comunicaciones OFDM mediante la implementación de los mismos. En este aspecto, podemos considerar que se alcanzado el objetivo previsto de forma rotunda.

Además de la puesta en marcha de la parte de transmisión de un sistema de comunicaciones, se ha realizado un estudio en profundidad sobre las librerías, manuales y configuraciones de los módulos del dispositivo SFF SDR (DSP, ADC/DAC, FPGA), que permitirá en el futuro, la continuación de diferentes proyectos de implementación.

Se han podido probar diferentes configuraciones hasta obtener unos resultados a la salida del dispositivo con un comportamiento muy similar al esperado teóricamente, en donde hemos podido analizar la transmisión de la señal en dos fases: con la señal separada en fase y cuadratura y con la señal modulada en banda RF. Mediante el análisis de estos resultados hemos podido comprobar el funcionamiento de un sistema transmisor OFDM, así como, posibles mejoras futuras en el sistema.

Asimismo, se ha podido trabajar en la familiarización de equipos de altas prestaciones en un laboratorio de comunicaciones que han permitido obtener unas mediciones y resultados con una alta precisión y calidad de imagen.

A su vez, se ha demostrado el camino utilizado cómo válido, mientras que las experiencias previas llevadas a cabo en el dispositivo, mostraban una gran ineficiencia en la implementación usando MATLAB y Simulink.

En este aspecto, se ha podido trabajar con el entorno de desarrollo centrado en la programación independiente de cada módulo del dispositivo SFF SDR, y consiguiendo la coordinación entre los diferentes módulos.

En cuanto a la implementación llevada a cabo en el presente proyecto, se han obtenido unos muy buenos resultados con respecto a la eficiencia de recursos. El hecho de haber llevado a cabo la programación del sistema de comunicaciones en el lenguaje VHDL, ha permitido una mejora en la utilización de las posibilidades ofrecidas por el módulo FPGA. Esta mejora supone liberar una parte de la capacidad del dispositivo SFF SDR lo que puede ser utilizado en líneas futuras para incrementar las operaciones del sistema. Al mismo tiempo, el uso del lenguaje VHDL, nos ha proporcionado la portabilidad buscada para otros módulos FPGA.

Respecto al desarrollo del sistema completo (modulador y demodulador), se ha demostrado mediante las simulaciones, el correcto funcionamiento de todos los bloques partícipes, obteniéndose a la salida del sistema, la misma información que había a la entrada del mismo.

Por último, tanto en el estudio realizado sobre los sistemas de comunicaciones OFDM, como en su posterior implementación, se ha profundizado en partes muy comunes en los sistemas OFDM, y con un mayor grado de complejidad que los bloques del esquema básico. En este aspecto, destaca el estudio sobre los algoritmos FFT y sus posibles implementaciones, y sobre la sincronización entre el transmisor y el receptor, en los que se han conseguido resultados óptimos.

En resumen, podemos afirmar que el presente proyecto ha cumplido con los objetivos propuestos, destacando dos de los puntos más importantes: se ha conseguido desarrollar y probar, con unos buenos resultados, una línea de trabajo para implementaciones hardware de sistemas de comunicaciones, y se ha implementado en un lenguaje programable y de bajo nivel como VHDL, un sistema de comunicaciones de gran interés como OFDM.

8 LÍNEAS FUTURAS

En este apartado se presentan posibles ampliaciones que se pueden llevar en la continuación del proyecto. Enumeramos y describimos brevemente las más importantes:

- **Implementación en la placa del receptor.**

  En el presente proyecto se ha implementado en el dispositivo SFF SDR el transmisor del sistema de comunicaciones OFDM. Para que la implementación sea del sistema completo, habría que implementar el demodulador desarrollado. Así como, configurar los bloques de conversión de datos y RF, para que, además de cómo funcionan en la actualidad, funcionen también para separar la señal recibida en banda, a dos señales, fase y cuadratura, y muestrearlas.

- **Nivel de aplicación**

  En el caso de implementar el receptor, una mejora, consistiría, en utilizar la información recibida (bits) a nivel de aplicación como podría ser una muestra de audio o una imagen. Para esto, habría que llevar a cabo una comunicación entre la interfaz que representa la información y el sistema de comunicaciones.

- **Algoritmos de sincronización**

  El bloque de sincronización del presente proyecto, se centra en la sincronización temporal que lleva a cabo el receptor en la detección del inicio de una trama OFDM recibida. Otras operaciones que mejoran la señal recibida y la recuperación de la información son el control automático de ganancia (AGC), la sincronización en frecuencia, la estimación de canal y la corrección de offset.

- **Número de portadoras**

  Durante la modulación de la información, se ha utilizado un número de portadoras fijo a 64. Este valor es mínimo para los algoritmos Radix-4 que calculan la FFT y la IFFT. Sin embargo, una posible ampliación sería probar los resultados con un número mayor de portadoras.

- **Tipo de modulación**

  En el desarrollo de este proyecto se ha trabajado con una constelación fija 4-QAM. Sin embargo, en ciertos módulos, se ha dejado la puerta abierta a futuros cambios en la constelación. Una posible ampliación podría ser el uso de diferentes constelaciones (16-QAM, 64-QAM, etc.).

- **Estudio de la BER y SNR**

  En el caso de implementar el receptor, se podrían obtener resultados que evalúen la BER y la SNR en función de parámetros como la constelación utilizada, el número de portadoras, e incluso, tratar de introducir distorsión del canal.

- **Cálculo de IFFT**

  Durante las simulaciones y la obtención de resultados, hemos visto como el tiempo entre trama y trama, es el que tarda el programa en transformar los bits de información en la modulación OFDM. El cuello de botella de este cálculo se encuentra en el modulador en el cálculo de la IFFT. Otra posible ampliación, sería reducir este tiempo de cálculo, implementando varios módulos que calculen la IFFT en paralelo.

- **Ajustar el filtro FIR del DAC**

  Durante la fase de resultados, vimos que teníamos un compromiso entre la calidad de la señal a la salida de módulo conversor de datos, y la eficiencia espectral de la señal. Tal vez, la obtención de un reloj ajustado a la frecuencia de trabajo mediante el ajuste del PLL interno o el uso de un reloj externo como frecuencia del DAC, puede suponer una mejora en las características de la señal OFDM analógica.

9 APÉNDICES

APÉNDICE A: PRESUPUESTO

En este apartado se detalla el presupuesto asociado a la realización del presente proyecto, contabilizando tanto los costes de personal de quien lo desarrolla, como el material empleado para la consecución del mismo.

A la suma de los costes directos de personal y material, se suman los costes indirectos generados durante el periodo de trabajo. Estos costes son imputables al lugar de trabajo (la universidad) y los estimaremos en torno al 20% de la suma de los costes de personal y material.

La suma de todos los costes constituye el presupuesto final del proyecto.

A.1 COSTES DE PERSONAL

El coste del personal está principalmente asociado al trabajo realizado por el ingeniero de telecomunicaciones encargado de la realización del proyecto. Tomamos como presupuesto de personal el sueldo bruto de un ingeniero de telecomunicación junior contratado por la universidad: 27.391,92 € al año.

Teniendo en cuenta el número de horas de trabajo efectivo de un trabajador ronda unas 1.575 horas laborables al año (descontando los días de vacaciones no laborables), obtenemos el coste por hora de personal.

27.391,92€ / 1.575 horas = 17,39 €/hora

El tiempo empleado en el desarrollo del proyecto lo estimamos en función de los meses que ha costado su ejecución, y las horas trabajadas. El tiempo empleado se corresponde al período comprendido entre Septiembre de 2009 y Octubre de 2010, es decir 12 meses, y se han empleado de media unas cuatro horas diarias.

Considerando que el número medio de días laborables en un mes es de 20 días, obtenemos una estimación de 960 horas. Con lo que el coste total del personal asciende **16.696,02€.**

Estos resultados quedan recogidos en la tabla A.1.

Personal	Salario Bruto	Horas empleadas	Honorarios (€/hora)	Importe
Ingeniero Junior	24,000	960	15,23	16.696,03
			Total	**16.696,03€**

Tabla A.1: *Costes de personal*

A.2 COSTES DE MATERIAL

En este apartado, detallamos los costes del material necesario para la ejecución del proyecto, valorando la vida útil del mismo.

Durante la realización del proyecto, comenzaron proyectos paralelos que también utilizarán los dispositivos usados en el presente proyecto. Sin embargo, a la finalización del mismo, estos proyectos paralelos se encontraban en fases previas al uso del dispositivo, por lo que no ha sido compartido el uso del material. De esta forma, los valores actualizados del coste de material sólo tendrán en cuenta su amortización en el tiempo.

Para el cálculo de la amortización se considerará una vida útil de todo el material de 5 años, lo que supone 60 meses.

La fórmula empleada para el cálculo de la amortización es la siguiente:

$$\frac{A}{B} \text{x} C$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)

(A.1)

Donde, en los costes de material, no se incluye el IVA(18%) ya que se incluye en la tasa de costes indirectos y no se amortiza.

En la tabla A.2 tenemos el cálculo de los costes de material que se componen de cuatro apartados. El primero es el dispositivo SFF SDR donde se ha llevado a cabo la implementación y que incluía las licencias software de “Xilinx ISE 9.1” y del “Lyrtech Development Tools”. Además de estas licencias, se ha necesitado licencia para el “Code Composer Studio v3.3” que vemos en la segunda fila.

En el tercer apartado se contabiliza osciloscopio del laboratorio de comunicaciones, las sondas que traía consigo y la garantía del mismo. Y el último apartado se corresponde con el ordenador de sobremesa empleado en la programación y cuyo valor con IVA se estima en 800€.

Descripción	Coste (€)	Dedicación (meses)	Periodo de depreciación	Coste imputable
SFF SDR DP	8.372,20	12	60	1.674,44
CCS 3.3	2.181,20	12	60	436,24
DSO90604A Osciloscopio Infiniium + Sondas + Garantía	41.967,45	12	60	8.393,49
Ordenador de sobremesa	656,00	12	60	131,20
			Total	**10.635,37€**

Tabla A.2: *Costes de Material*

A.3 COSTES TOTALES

Sumando los costes de personal, los costes de material y los costes indirectos generados (20% de la suma de los costes de personal y de material) obtenemos el presupuesto total del presente proyecto, que podemos observar en la tabla A.3.

Concepto	Presupuesto (€)
Costes de personal	16.696,03
Costes de material	10.635,37
Costes indirectos	5466,28
Total	**32.797,68€**

Tabla A.3: *Presupuesto total*

APÉNDICE B: GUÍA PARA GENERAR Y CARGAR PROGRAMAS EN EL DISPOSITIVO SFF SDR

En este apartado se detallar el proceso que hay que llevar a cabo para comunicarse con el dispositivo SFF SDR y para generar y cargar los programas ejecutables.

B.1 GENERACIÓN DE ARCHIVOS

En primer lugar, tenemos que generar los archivos bit y out desde el Xilinx ISE y el Code Composer Studio, respectivamente.

Para generar el archivo bit con el código programable en la FPGA tenemos que ejecutar todos los pasos posibles en el Xilinx ISE: sintetizar, implementar y programar (figura B.1).

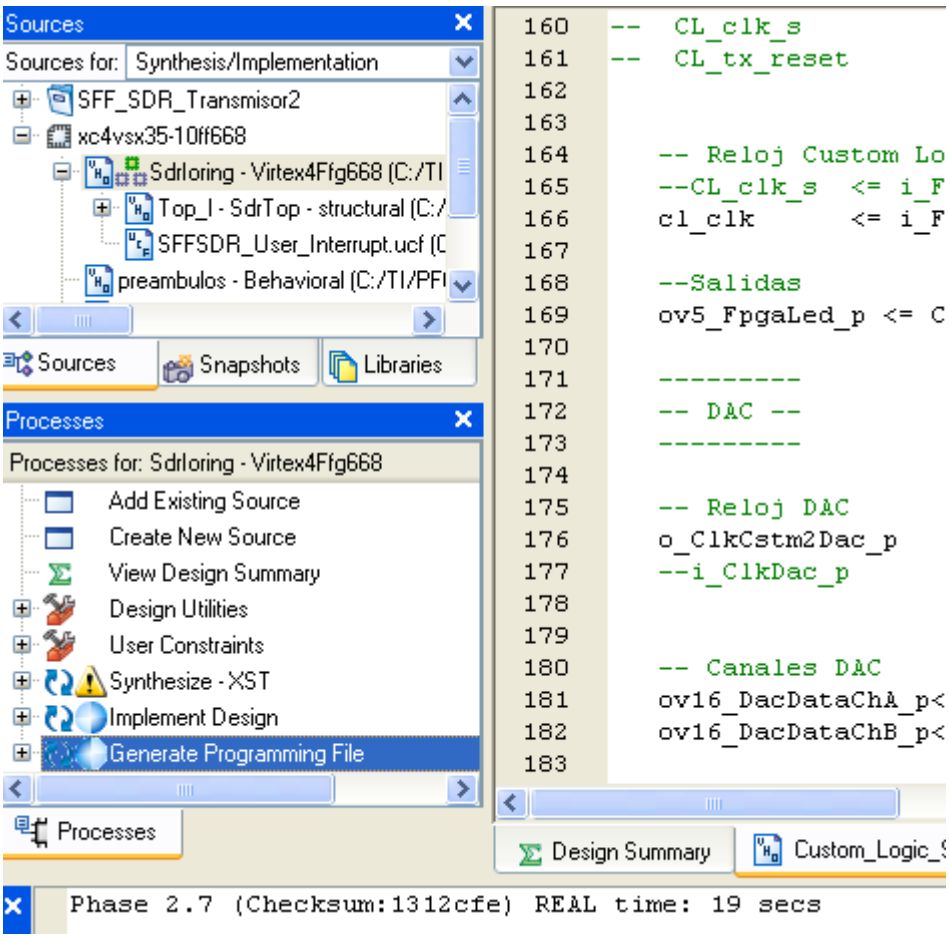


Figura B.1: *Proceso de generación del archivo programable bit*

Si no hay ningún fallo, el programa nos muestra el éxito del proceso, como vemos en la figura B.2.

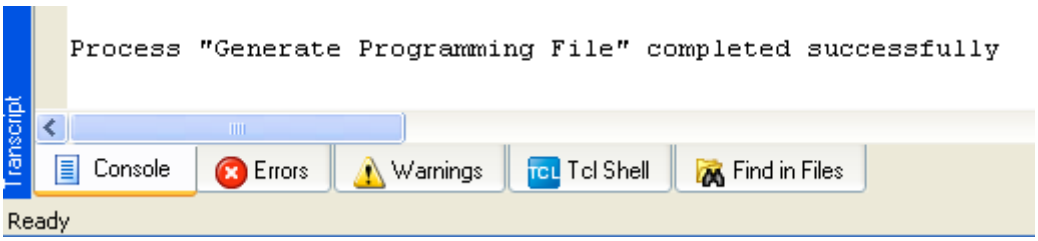


Figura B.2: *Proceso de generación del archivo programable bit finalizado con éxito*

Para generar el archivo out programable en el DSP, se ejecuta la acción “Build” o “Rebuild All” (figura B.3).

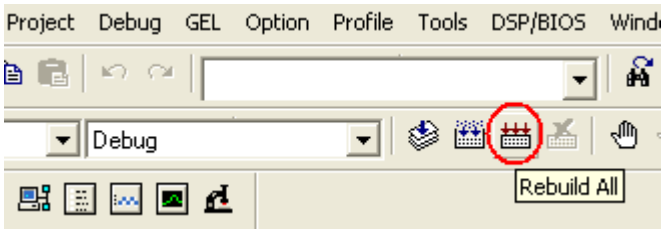


Figura B.3: *Ejecución de generación del archivo programable out*

En la figura B.4 vemos como el programa nos informa de que la generación del archivo ha tenido éxito.

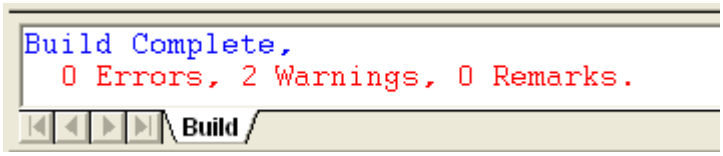


Figura B.4: *Éxito en la generación del archivo programable out*

La generación de ambos archivos se encontrará en el directorio de trabajo de cada proyecto respectivamente.

B.2 CARGA DE ARCHIVOS

Para poder cargar los archivos creados en el dispositivo SFF SDR, hay que configurar la comunicación con el dispositivo. La forma más sencilla es utilizar un cable de red con conectores RJ-45, para conectar directamente el dispositivo al ordenador.

La placa viene por defecto configurada a la IP 10.0.0.2 (Máscara de subred 255.255.255.0). Por lo que para conseguir comunicar, la dirección IP de la interfaz donde se conecta el cable de red del ordenador, debe de tener asignada una IP dentro del rango 10.0.0.XXX. Es decir, un ejemplo es configurar esta interfaz a los siguientes valores:

- Dirección IP: 10.0.0.1
- Máscara de subred: 255.255.255.0

En el caso de que el sistema operativo no detecte la conexión establecida, se recomienda seguir los pasos dados en los manuales [35] y [33], para modificar la IP del dispositivo a un valor conocido.

A continuación, tras conectar el cable, encender el dispositivo y que el sistema operativo lo detectar, arrancamos el "Command Shell" perteneciente al "Lyrtech Development Software Tools".

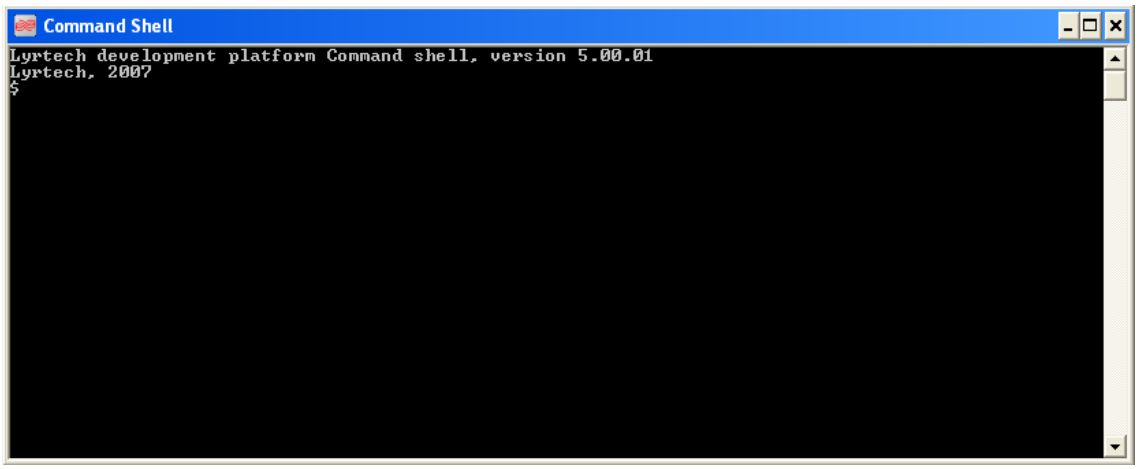


Figura B.5: *Command Shell*

Siguiendo las instrucciones del manual [17], ejecutamos la instrucción *"smdetect".*

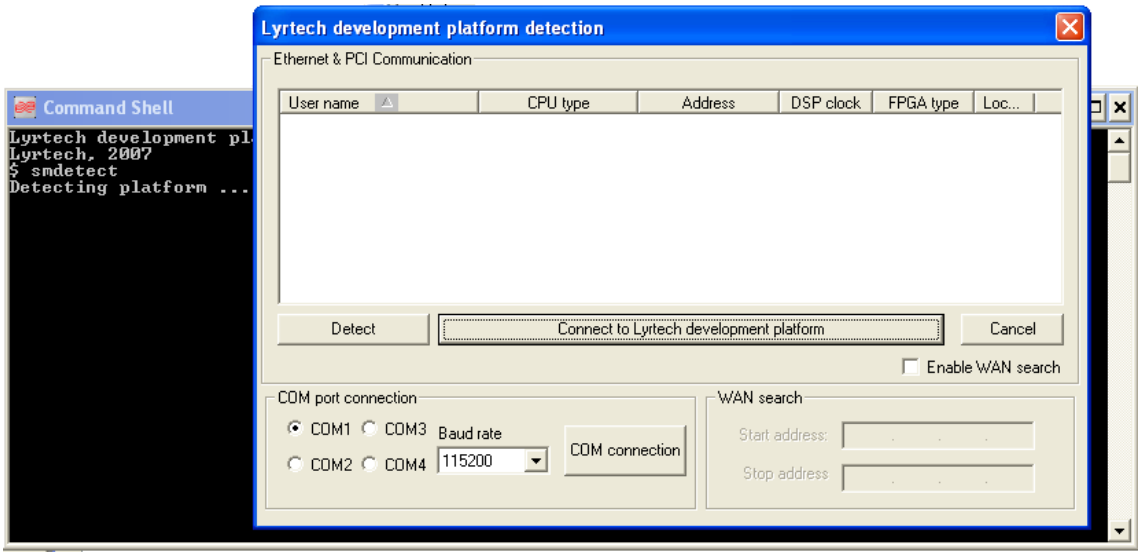


Figura B.6: *Arranque de Lyrtech development plattform detection*

Al ejecutar “smdetect”, se ejecuta el programa “*Lyrtech development plattform detection*”. Para que este programa encuentre el dispositivo, es recomendable darle un rango de IPs en el que buscarlo, como vemos en la figura B.7. Una vez dado el rango de IPs, pulsaremos sobre “Connect to Lyrtech development plattform” y obtendremos los datos tanto de la DSP como de la FPGA programables y pertenecientes al dispositivo SFF SDR.

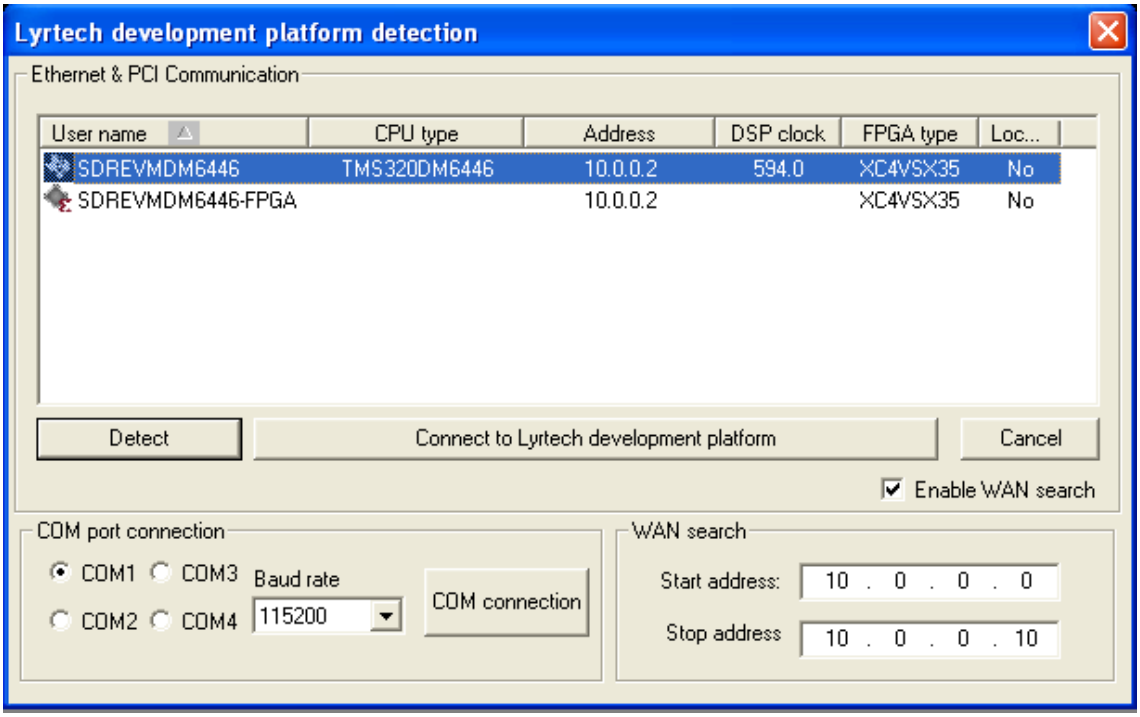


Figura B.7: *Lyrtech development plattform detection*

Pinchamos dos veces sobre SDREVMDM6446, que nos conectará con el dispositivo, volviendo al “*Command Shell*”. Si los pasos se han hecho correctamente, nos aparecerá la plataforma bloqueada “*Platform locked*”(figura B.8)

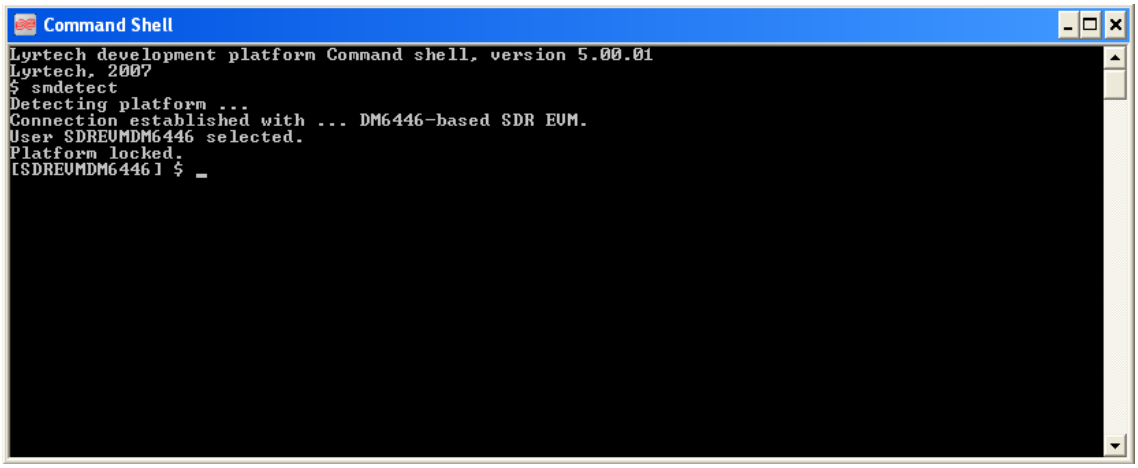


Figura B.8: *Conexión establecida en el Command Shell*

En este punto, ya podemos cargar los archivos ejecutables en el dispositivo. Comenzamos cargando el archivo bit en la FPGA mediante el comando FPGALOAD.

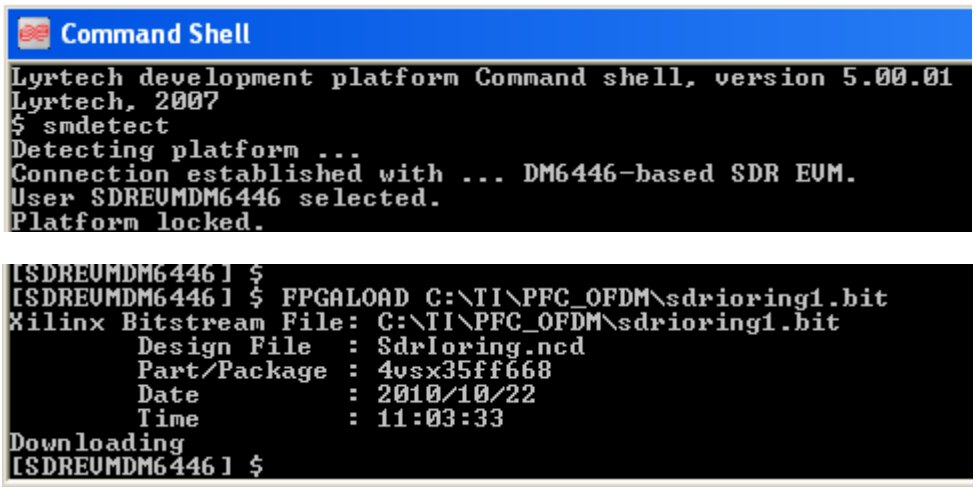


Figura B.9: *Carga del fichero bit en la FPGA*

Si obtenemos una pantalla como la captura de la figura B.9, significa que no ha habido ningún problema. Procedemos de manera similar a cargar el archivo out en la DSP, utilizando en este caso el comando DSPLOAD.

Figura B.10: *Carga del fichero out en el DSP*

Con lo que en este momento, el dispositivo está programado y configurado con nuestro programa.

GLOSARIO

ADC	Analog to Digital Converter
AGC	Automatic Control Gain
BER	Bit Error Ratio
BSDK	Board Software Development Kit
BUFG	Global Buffer
CCS	Code Composer Studio
CLB	Configurable Logic Block
CP	Cyclic Prefix
DAB	Digital Audio Broadcasting
DAC	Digital to Analog Converter
DCM	Digital Clock Manager
DFT	Discrete Fourier Transform
DIF	Decimation in Frequency
DIT	Decimation in Time
DNS	Domain Name System
DSP	Digital Signal Processor
DVB-T	Digital Video Broadcasting – Terrestrial
EEPROM	Electrically-Erasable Programmable Read-Only Memory
FDM	Frequency-Division Multiplexing
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
GPP	General-Purpose Processor
ICI	Inter Carrier Interference
IEEE	Institute of Electrical and Electronics Engineers
IF	Intermediate Frequency
IFFT	Inverse Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
IOB	Input Output Blocks
IP	Internet Protocol
ISI	Inter Symbol Interference
JTAG	Joint Test Action Group
LAN	Local Area Network
LP	Low Pass
LTE	*Long Term Evolution*

LUT	*Lookup Table*
MBDK	Model-Based Development Kit
MSPS	Mega Sample Per Second
OFDM	Orthogonal Frequency-Division Multiplexing
OFDMA	Orthogonal Frequency-Division Multiple Access
PLCP	Physical layer Convergence Procedure
PLL	Phase Locked Loop
PPDU	PLCP Protocol Data Unit
PSDU	Physical layer Service Data Unit
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
RF	Radio Frequency
SDR	Software Defined Radio
SFF	Small Form Factor
SMA	SubMiniature version A (Coaxial Connector)
SNR	Signal to Noise Ratio
SoC	System on Chip
VCO	Voltage Controlled Oscillator
VPSS	Video Processing SubSystem
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
XST	Xilinx Synthesis Tools

REFERENCIAS

[1] Roberto Corvaja, Ana Garcia Armada, "Joint Channel and Phase Noise Compensation for OFDM in Fast Fading Multipath Applications". IEEE Transactions on Vehicular Technology, Vol. 58, no. 2, pp. 636-643, February 2009

[2] Víctor P. Gil Jiménez, M. Julia Fernández-Getino García, Francisco J. González Serrano, and Ana García Armada. "Design and Implementation of Synchronization and AGC for OFDM-based WLAN Receivers". IEEE Transactions on Consumer Electronics, Vol. 50, No. 4, Noviembre de 2004

[3] David Díaz Martín. "Prototipado de un sistema WiMaX MIMO 2x2" PFC-UC3M. 2010

[4] Tomás Alemany Sánchez. "Descripción Hardware de Algoritmos de estimación de canal y sincronización en tiempo-frecuencia para un sistema 2x2 MIMO-OFDM" PFC-UC3M. Octubre de 2010

[5] Lyrtech Inc. "SFF SDR Development Platform User's guide" version 1.1, Octubre de 2007.

[6] Texas Instruments Inc. Página web principal de Texas Instruments. Disponible en http://www.ti.com/

[7] Texas Instruments Inc. Página web de Texas Instruments "Embedded Processors Wiki". Disponible en http://processors.wiki.ti.com

[8] Xilinx Inc. Página web principal de Xilinx. Disponible en http://www.xilinx.com/

[9] Lyrtech Inc. Página web principal de Lyrtecht. Disponible en http://www.lyrtech.com/

[10] Datasheet del DAC5687 de Texas Instruments. Disponible en http://focus.ti.com/lit/ds/symlink/dac5689.pdf Agosto 2009

[11] Virtex 4 FPGA User Guide, Xilinx, December 2008.

[12] TMS320DM6446 Digital Media System-on-Chip Data Sheet, Texas Instruments. Marzo de 2008.

[13] Miguel Ángel Freire Rubio "Introducción al lenguaje VHDL". UPM-Dep. de Sistemas Electrónicos y de Control. Febrero de 2008

[14] Douglas L.Perry "VHDL: Programming by Example". McGraw-Hill. 2002

[15] Xilinx Inc. "Xilinx ISE In-Depth Tutorial". 2007

[16] Xilinx Inc. "Virtex-4 Libraries Guide for HDL Designs". 2007

[17] Texas Instruments Inc. "Code Composer Studio Development Tools v3.3. Getting Started Guide". Octubre de 2007

[18] Lyrtech Inc. “Lyrtech Development Platform Command Shell - User’s Guide” version 1.6. Octubre de 2007

[19] Antonio Artés Rodríguez. “Comunicaciones Digitales”. Pearson COLLEGE. 2007. 775p. ISBN: 848322348. ISBN 13: 9788483223482

[20] Charan Langton. “Intuitive Guide to Principles of Communications”. Disponible en http://www.complextoreal.com Octubre de 2002.

[21] Bahai, Ahmad R.S. “Multi-carrier digital communications : theory and applications of OFDM”. Springer 2004. ISBN: 0387225757

[22] Página web “Mobile & Wireless - An Overview of OFDM” Disponible en http://mobilewireless.wordpress.com

[23] Cesar V. Vargas, Wilson E. Lopez, Carlos F. da Rocha. “Sistemas de Comunicación Inalámbrica MIMO - OFDM”. UFSC. Junio de 2007

[24] Wireless Communication. “Analog and Digital Transmission”. Disponible en: http://www.wirelesscommunication.nl/reference/contents.htm#transmission 2006

[25] Marco Antonio Muñoz Valdebenito. “Metodologías, criterios y herramientas para la planificación de redes inalámbricas”. UNIVERSIDAD DE CHILE. Mayo de 2007.

[26] Página Web de Marcus Winter. “Optical OFDM”. Disponible en http://www.marcuswinter.de/archives/884. Agosto de 2010

[27] IEEE Std 802.11a-1999. “Standard for Information technology: 802.11a - Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”. 1999

[28] Pere Martí, David Novo, Moisès Serra, Juli Ordeix, Jordi Carrabina. “Diseño de un bloque FFT/IFFT con alto grado de paralelismo mediante FPGA”. Jornades de Codsseny Hardware Software, Vic, 26 y 27 de Junio 2003.

[29] P. Sonna, E. Serrano, L. Castillo. “Procesador Paralelo de FFT Implementado en FPGA”, XIII Workshop Iberchip. 2007

[30] Xilinx Inc. “Fast Fourier Transform v3.1 Product Specification”. Noviembre de 2004

[31] Amontec Inc. “Using Xilinx IP cores” version 0.1. Junio de 2001

[32] Lyrtech Inc. “SFF SDR DP API Guide” version 1.0, Enero de 2007

[33] Lyrtech Inc. “Lyrtech Quick start Guide” version 1.5. Octubre de 2007

[34] Stefan Nagel, Michael Schwall, Friedrich K. Jondral. “Model Based Implementation of SDR Waveforms”. Karlsruhe Institute of Technology (KIT),Communications Engineering Lab (CEL), Germany. 2010.

[35] Green Hills Software, Inc. “Lyrtech SFF SDR EVM-DP Network Configuration” version 4.3.1. Octubre de 2007.

[33] Wikipedia. Disponible en http://es.wikipedia.org/. Septiembre de 2010.